Design Report

Gas Turbine Design

April 24, 2024

Department of Mechanical, Industrial and Aerospace Engineering,
Concordia University
Montreal, QC, Canada

Technical Report

Core Engine Design

Date: April 24, 2024

AeroSpec
Reliable Engines

Collaborative Innovation Department

Address: 1455 Blvd. De Maisonneuve Ouest
Montreal, Quebec H3G 1M8

Contact: info@aerospec.com

This report is submitted in partial fulfillment of the requirements for the partnership agreement with Pratt & Whitney Canada.

Contents

List of Figures

List of Tables

Nomenclature - Compressor

N_s	-	Centrifugal Specific Speed
ρ	-	Density
Q	-	Volume Flow
$\Delta h_{0,\text{is}}$	-	Isentropic Enthalpy Rise
N	-	Rotational Speed
w	-	Rotational Speed
r	-	Radius
L	-	Compressor Length
b	-	Height
N_b	-	Number of Blade
T	-	Temperature
$\dot{m}$	-	Mass Flow Rate
C	-	Absolute Velocity
α	-	Swirl Angle
U	-	Blade's Velocity
W	-	Relative Velocity
β	-	Back Sweep Angle
M_{rel}	-	Relative Mach Number
θ	-	Flow Turning Angle
W_{thermo}	-	Thermo Work
σ	-	Slip Factor
A	-	Area
i	-	Incidence Angle
β^*	-	Metal's Angle
P	-	Pressure

Subscripts

1	-	At the Impeller's Inlet
2	-	At Impeller's Outlet, At the Diffuser's Inlet
3	-	At Diffuser's Outlet
h	-	At the Hub
m	-	At the Mean
sh	-	At the Shroud
$partA$	-	From Part A
x	-	Axial's Direction
r	-	Axial's Direction

Nomenclature - Turbine

h	-	Blade Height
c_{true}	-	True Chord
c_a	-	Axial Chord
C_L	-	Lift Coefficient
f_{AR}	-	Aspect Ratio Function
$K_1, K_2, K_3, K_p^*, K_{sh}, K_s^*, K_{accl}$	-	Correction Loss Factors
K_T, K_p, K_s, K_{TE}	-	Loss Coefficients
s	-	Pitch
o	-	Throat Opening
Φ	-	Stagger Angle
R	-	Reaction
ψ	-	Stage Loading
ϕ	-	Flow Coefficient
$U_{mean,hub,tip}$	-	Blade Speed at Mean, Hub, Tip
AN^2	-	Annulus Area x Rotational Speed Squared
ω	-	Angular Velocity
ρ	-	Density
$P_{0,rel}$	-	Relative Stagnation Pressure
M_{hub}	-	Absolute Mach Number at Hub
M_{rel}	-	Relative Mach Number
M_{abs}	-	Absolute Mach Number
T_0	-	Stagnation Temperature
P_0	-	Stagnation Pressure
$\dot{m}$	-	Mass Flow Rate
P	-	Static Pressure
T	-	Static Temperature
C	-	Absolute Velocity
C_w	-	Tangential Velocity
C_a	-	Axial Velocity
α	-	Absolute/Relative Flow Angle
β	-	Metal Angle
V	-	Relative Velocity
V_w	-	Tangential Relative Velocity
A	-	Area
$r_{hub,mean,tip}$	-	Radius at Hub, Meanline, Tip
η_{tt}	-	Total-to-Total Turbine Efficiency
η_o	-	Efficiency at 0 Tip Clearance
N	-	Number of Vanes/Blades

Section 1

Cycle Calculations

The objective of this technical report is to present a design for a new family of gas turbine engines under a risk-sharing partnership with Pratt & Whitney Canada. The turboshaft engine design should have a three shaft arrangement for sea level operation at 74^oF. The engine features a single-stage low-pressure compressor (LPC) and axial turbine (LPT), followed by a centrifugal high-pressure compressor (HPC) and axial high-pressure turbine (HPT). The power turbine (PT) drives the helicopter rotor through a reduction gearbox.

The following figure presents the engine schematics along with all the stations.

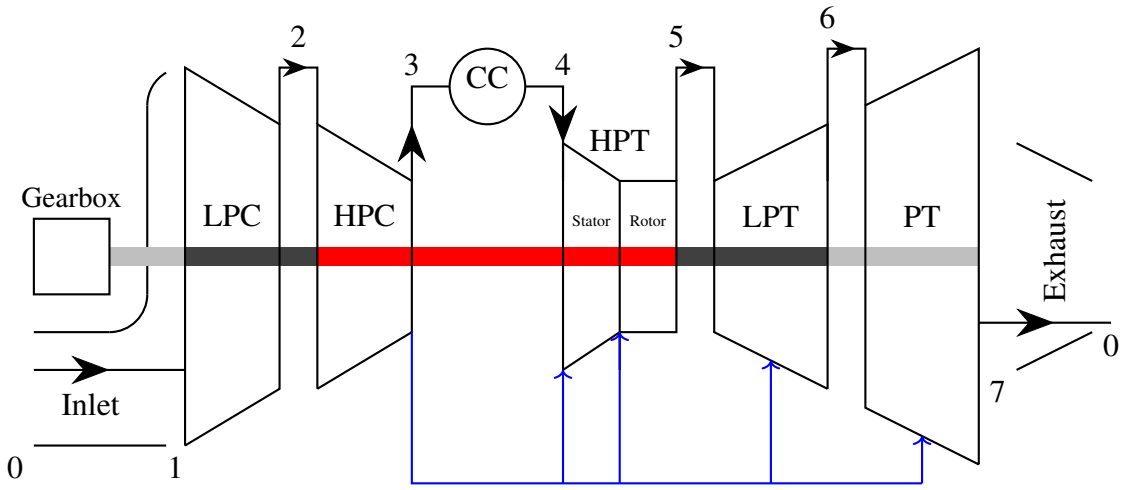


Figure 1.1: Engine Schematic

The key engine requirements are summarized in Table 1.1.

Table 1.1: Engine Characteristics

Engine Inlet		*Compressor*	
Inlet Loss	1.00 %	Mass Flow	5.21 kg/sec
		LPC PR	4
		LPC Target Efficiency	86.5 %
		HPC PR	3
		HPC Target Efficiency	85.5 %
		HPC Bleed Air (exit)	9 %
Combustor		*Turbine*	
Fuel to Air ratio	0.02	HPT Target Efficiency	83 %-85 %
Heating Value	40007.2 kJ/kg	HPT Vane Cooling Air	3 %
Efficiency	0.99	HPT Disk Cooling Air	1.65 %
Pressure Loss	1.8 %	LPT Target Efficiency	91 %
RTDF	5 %	LPT Disk Cooling Air	1.1 %
		ITD loss	0.6 %
		PT Target Efficiency	92 %
		PT disk cooling air	1.25 %
		Exhaust Loss	2.0 %
		Exhaust Mach Number	0.15

This section discusses the method used for the cycle calculations. The calculations for each stage are presented as well.

1.1 Inlet

Since $P_{00} = P_a$ and $T_{00} = T_a$, the pressure drop across the inlet can be calculated as:

$$P_{01} = 0.99 \times P_{00} = 100.311\ kPa$$

$$T_{01} = 296.483\ K$$

1.2 Low Pressure Compressor

The pressure at the LPT exit (P_{02}) can be calculated as:

$$P_{02} = PR_{LPC} \times P_{01} = 401.247\ kPa$$

The temperature can be obtained using the following relation:

$$T_{02} - T_{01} = \frac{T_{01}}{\eta_{lpc}} \left[\left(\frac{P_{02}}{P_{01}} \right)^{(\gamma-1)/\gamma} - 1 \right] \tag{1.1}$$

$$T_{02} = 463.059\ K$$

Therefore, the specific work done for the LPC:

$$W_{LPC} = c_{p,air} \times (T_{02} - T_{01}) = 167.409\ kJ/kg$$

1.3 High Pressure Compressor

Following the steps from Section 1.2, P_{03}, T_{03} and W_{HPC} can be calculated:

$$P_{03} = PR_{HPC} \times P_{02} = 1203.741\ kPa$$

$$T_{03} = 662.764\ K$$

$$W_{HPC} = c_{p,air} \times (T_{03} - T_{02}) = 200.703\ kJ/kg$$

Since the bleed air is obtained from the HPC exit, the resulting mass flow rate entering the combustion chamber:

$$\dot{m}_{combustion} = 0.91 \times \dot{m}_{total} = 4.741\ kg/s$$

1.4 Combustion Chamber

Pressure at the exit of the combustion chamber can be calculated using the combustion chamber pressure loss:

$$P_{04} = P_{03} \times 0.982 = 1182.073\ kPa$$

The temperature at the exit of the combustion chamber can be calculated using the following relation:

$$T_{04} = \frac{c_{p,air} \times T_{03} + f \times Q \times \eta_{cc}}{(1+f) \times c_{p,gas}} = 1245.320\ K$$

1.5 High Pressure Turbine

The following points are noted for the calculations of the turbine temperatures and pressures with cooling [4]:

- Turbine air cooling percentage considered as the percent flow of turbine inlet flow.
- Stator bleed contributes to power developed.
- Disc bleed do not contribute to power developed.

The combusted gas entering the high pressure turbine also included the mass of the fuel. This is calculated as:

$$\dot{m}_{HPT} = \dot{m}_{combustion} + 0.02 \times \dot{m}_{combustion} = 4.835\ kg/s$$

Similarly, the mass flow of air to be used for vane and disc cooling for the HPT is as follows:

$$\dot{m}_{vane,HPT} = \dot{m}_{turbine} \times 0.03 = 0.1450\ kg/s$$

$$\dot{m}_{disc,HPT} = \dot{m}_{turbine} \times 0.0165 = 0.0797\ kg/s$$

To obtain the temperature after the vane, the following energy balance can be performed:

$$T_{\text{hpt after vane}} = \frac{(\dot{m}_{\text{turbine}} \cdot c_{p_{\text{gas}}} \cdot T_{04}) + (\dot{m}_{vane,HPT} \cdot c_{p_{\text{air}}} \cdot T_{03})}{c_{p_{\text{gas}}} \cdot (\dot{m}_{\text{turbine}} + \dot{m}_{vane,HPT})}$$

$$T_{\text{hpt after vane}} = 1225.9485\ K$$

The temperature after the rotor can be evaluated based on the power required by the HPC:

$$T_{\text{hpt after rotor}} = T_{04} - \frac{1.01 \cdot W_{\text{hpc}}}{(m_{\text{turbine}} + \dot{m}_{vane,HPT}) \cdot c_{p,gas}} = 1041.2535\ K$$

The HPT exit temperature can be calculated by accounting for the disc bleed air:

$$T_{05} = \frac{((\dot{m}_{\text{turbine}} + \dot{m}_{vane,HPT}) \cdot c_{p_{\text{gas}}} \cdot T_{\text{hpt after rotor}}) + (\dot{m}_{disc,HPT} \cdot c_{p_{\text{air}}} \cdot T_{03})}{c_{p_{\text{gas}}} \cdot (\dot{m}_{\text{turbine}} + \dot{m}_{vane,HPT} + \dot{m}_{disc,HPT})}$$

$$T_{05} = 1033.9843\ K$$

The pressure can be evaluated using the isentropic efficiency. For this design, the HPT efficient has been selected as 84 %. The following relation is used:

$$T_{04} - T_{05} = \eta_{HPT} T_{04} \left[1 - \left(\frac{1}{P_{04}/P_{05}} \right)^{(\gamma-1)/\gamma} \right] \tag{1.2}$$

$$P_{05} = 517.9223\ kPa$$

1.6 Low Pressure Turbine

The total mass of the gas entering the LPT includes the cooling air from the HPT as well:

$$\dot{m}_{LPT} = \dot{m}_{HPT} + \dot{m}_{disc,HPT}$$

$$\dot{m}_{disc,HPT} = 0.011 \times \dot{m}_{turbine}$$

Following similar calculation procedure as seen in Section 1.5, the following values are calculated:

$$T_{\text{lpt after rotor}} = 882.3564\ K$$

$$T_{06} = 879.2135\ K$$

1.7 Exhaust

Since the exit conditions are already given in the problem statement, the calculations for the exhaust are carried out first. Using the exit loss and the exit mach number, the exit total pressure can be calculated using the following relation:

$$\frac{P_{08}}{P_0} = \left[1 + \left(\frac{\gamma - 1}{2}\right) M_{exit}^2\right]^{\frac{\gamma}{\gamma-1}} \tag{1.3}$$

$$P_{08} = 102.853\ kPa$$

Therefore, the Power Turbine exit pressure can be calculated as:

$$P_{07} = 1.02 \times P_{08} = 104.910\ kPa$$

1.8 Power Turbine

The total mass of the gas entering the PT includes the cooling air from the HPT and LPT:

$$\dot{m}_{PT} = \dot{m}_{LPT} + \dot{m}_{disc,LPT}$$

Incorporating ITD losses:

$$P_{06,PT} = (1 - ITD_{loss}) \times P_{06} = 240.755\ kPa$$

Following similar calculation procedure from Section 1.5:

$$T_{\text{pt after rotor}} = 727.5310\ K$$

After adding disc cooling air:

$$T_{07} = 725.8100\ K$$

1.9 Work and SFC

The work for the HPT can be computed as following.

$$W_{HPT} = c_{p,gas}(T_{06} - T_{07}) = 212.0kJ/kg$$

Work can be calculated as follows:

$$W_{PT} = c_{p,gas}(T_{06} - T_{07}) \times 0.99 = 172.4\ kJ/kg$$

$$SFC = \frac{3600 \times 0.02}{W_{PT}} = 0.4176$$

Section 2

High Pressure Compressor Design

In this section, the explanation of the High Pressure Compressor (HPC) design for the engine is provided. The HPC's integration with the High Pressure Turbine (HPT) is pivotal for the engine's operation. Employing an iterative method, the HPC's design was optimized, taking into account factors such as pressures, temperatures, and cycle calculations. Figure 2.1 depicts the centrifugal compressor's configuration.

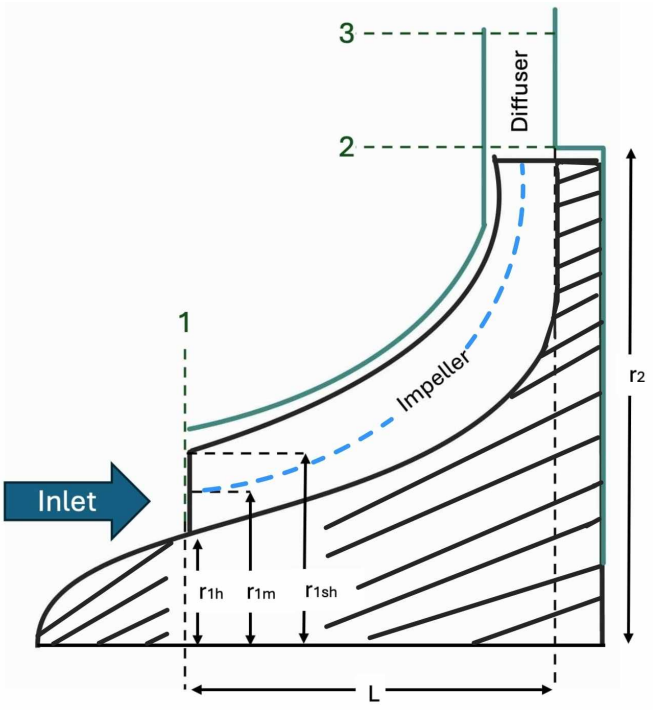


Figure 2.1: High Pressure Compressor Stages [5]

2.1 Impeller Inlet Design Parameters for Geometry

The rotational speed equations, presented in Table 2.1, were derived from the velocity triangle depicted in Figure 2.2. These equations underwent iterative adjustments of density to ensure alignment with the

desired rotational speed of the HPT. A suitable value for the specific speed (N_s) was selected within the range of 55 to 85 to minimize losses effectively [5].

Table 2.1: Rotational Speed [5]

Step	Variable	Equation	Value
0.01	Centrifugal Specific Speed	N_s was iterated	76.23
0.02	Density	ρ_1 was iterated	$2.950\frac{kg}{m^3} = 0.08355\frac{kg}{ft^3}$
0.03	Volume Flow	$Q = \frac{m_{03}}{\rho_1}$	$62.44\frac{ft^3}{s}$
0.04	Isentropic Enthalpy Rise	$\Delta h_{0,\text{is}} = c_{\text{pa}} \cdot (T_{04} - T_{03})$	$200.7\frac{kJ}{kg} = 86.29\frac{Btu}{lb}$
0.05	Rotational Speed	$N = \frac{(N_s(778.26 \cdot \Delta h_{0,\text{is}}))^{3/4}}{\sqrt{Q}}$	$N = 40,245 rpm; w = 4214\frac{rad}{s}$

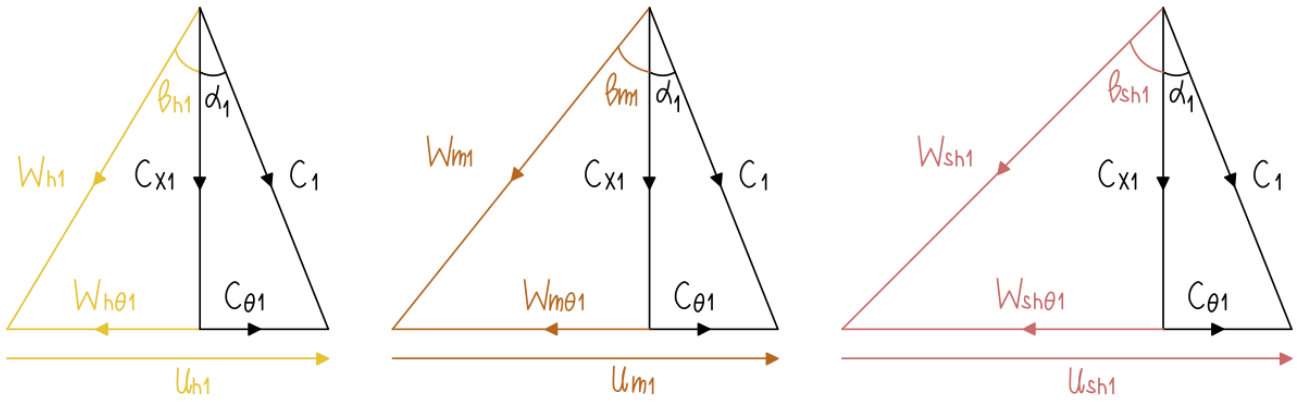


Figure 2.2: Impeller Inlet Velocity Triangle [5]

Similarly, the compressor's design formulas are outlined in Table 2.2 and were derived from the principles illustrated in Figure 2.3 [5]. Moreover, Figure 2.1 provides examples of the radius configurations. Notably, there are fewer blades at the leading edge of the impeller compared to the trailing edge, owing to the presence of 16 splitter blades.

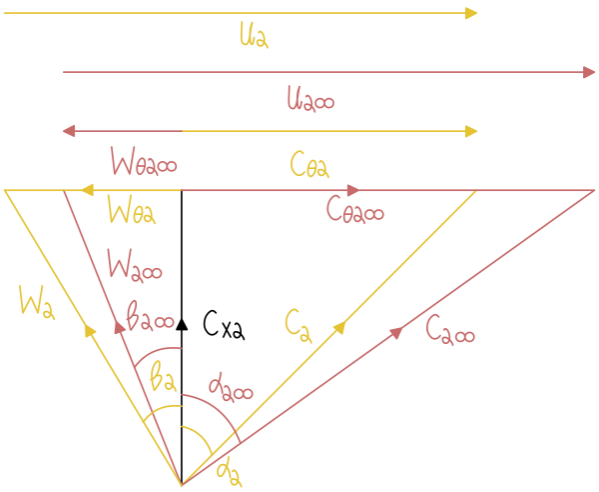


Figure 2.3: Impeller Exit Velocity Triangle [5]

Table 2.2: Compressor's Dimensions [5]

Step	Variable	Equation	Value
0.06	Impeller Hub Leading Edge Radius	r_{1h} was iterated	$0.075m$
0.07	Impeller Mean Leading Edge Radius	$r_{1m} = \frac{r_{1h}+r_{1sh}}{2}$	$0.085m$
0.08	Impeller Shroud Leading Edge Radius	r_{1sh} was iterated	$0.095m$
0.09	Impeller Trailing Edge Radius	r_2 was iterated	$0.125m$
0.10	Radius Ratio	$1.05 \leq \frac{r_3}{r_2} \leq 1.10$	1.1
0.11	Pressure Ratio	$0.9 \leq \frac{r_2-r_{1h}}{L} \leq 1.2$	1.2
0.12	Diffuser Trailing Edge Radius	$r_3 = r_2 \cdot \frac{r_3}{r_2}$	$0.1375m$
0.13	Compressor Length	$L = \frac{r_2-r_{1h}}{(\frac{r_2-r_{1h}}{L})}$	$0.04167m$
0.14	Etha max	e_{max} was given	0.01
0.15	Impeller Exit Channel Height	$b_2 = \frac{C_{r2} \cdot A_2}{C_2 \cdot 2\pi \cdot r_2 (1-B^*_{2aero}-B^*_{2blade})}$	$0.00716m$
0.16	Trailing Edge Diffuser Height	$b_3 = b_2$	$0.00716m$
0.17	Channel Height	$b_{2-3} = b_2$	$0.00716m$
0.18	Leading Edge Number of Blade	N_{b1} was given	16
0.19	Trailing Edge Number of Blade	N_{b2} was given	32

Table 2.3 shows the incidence's angles assumed for the compressor's design.

Table 2.3: Incidence's Angles [5]

Step	Variable	Equation	Value
1.01	Inlet Shroud Incidence's Angle	$1° \leq i_{1sh} \leq 3°$	2°
1.02	Inlet Mean Incidence's Angle	$2° \leq i_{1m} \leq 5°$	4°
1.03	Inlet Hub Incidence's Angle	$5° \leq i_{1h} \leq 9°$	7°

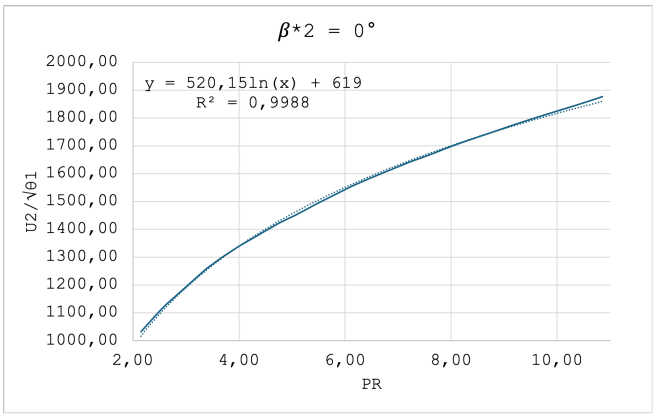
The formulas utilized to derive the parameters and values for the first stage are displayed in Table 2.4. Iterating the inlet swirl angle is crucial to achieving a high level of efficiency. Furthermore, shock waves are not expected, it's crucial that all of the Mach numbers are less than 1.

2.2 Impeller Exit Design Parameters for Geometry

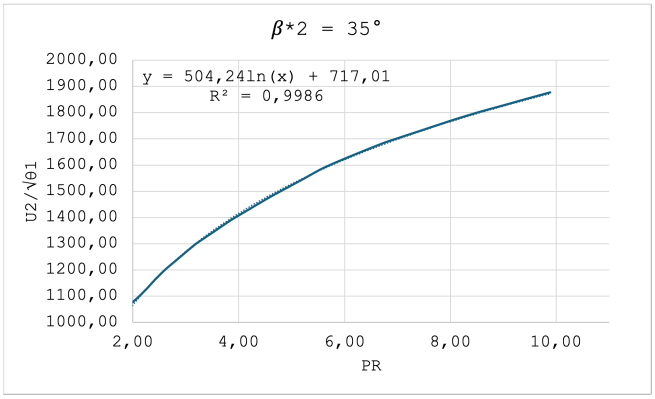
The plots to get the U_2 corrected at various values of B_2^* are Figure 2.4a, Figure 2.4b, and Figure 2.4c. Furthermore, the exit design parameter values and equations are displayed in Table 2.5. To get a good efficiency, the exit back sweep angle and the exit swirl angle were iterated values. In Step 2.14, the relative velocity ratio needed to be between 0.5 and 0.6 in order to avoid an overly aggressive design. As well as that, for the design to be considered, the meridional velocity ratio needed to fall between 0.8 and 1.0. Lastly, there needs to be convergence between the start and exit back sweep angle verifications.

Table 2.4: Stage 1 [5]

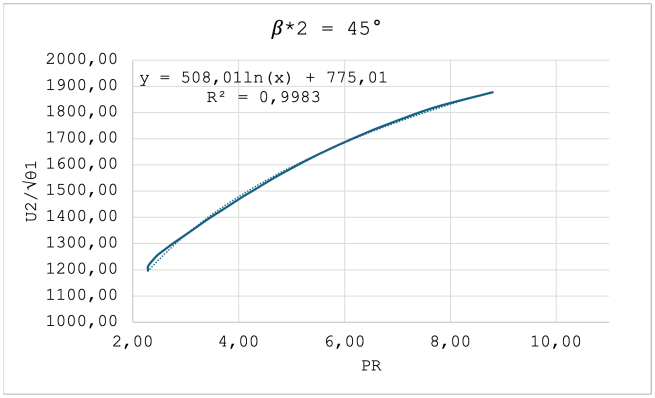
Step	Variable	Equation	Value
1.04	Inlet Temperature	$T_{01} = T_{03:PartA}$	$463.1K$
1.05	Mass Flow Rate	$\dot{m}_1 = \dot{m}_{02}$	$5.216\frac{kg}{s}$
1.06	Eye Are	$\pi({r_{1sh}}^2 - {r_{1h}}^2)$	$0.011m^2$
1.07	Axial Velocity	$C_{x1} = \frac{\dot{m}_1}{\rho \cdot A_{1sh}}$	$165.5\frac{m}{s}$
1.08	Inlet Swirl	$5° \leq \alpha_1 \leq 35°$	$20° = 0.3491rad$
1.09	Absolute Velocity	$C_1 = \frac{C_{x1}}{cos(\alpha_1)}$	$176.1\frac{m}{s}$
1.10	Horizontal Absolute Velocity	$C_{\theta 1} = \frac{C_{x1}}{tan(\alpha_1)}$	$60.24\frac{m}{s}$
1.11	Blade Velocity at Tip	$U_{1sh} = w \cdot r_{1sh}$	$400.4\frac{m}{s}$
1.12	Horizontal Relative Velocity at Tip	$W_{\theta 1sh} = U_{1sh} - C_{\theta 1}$	$340.1\frac{m}{s}$
1.13	Inlet Backsweep Angle at Tip	$\beta_{1sh} = tan^{-1}(\frac{W_{\theta 1sh}}{C_{x1}})$	$1.118rad = 64°$
1.14	Relative Velocity at Tip	$W_{1sh} = \sqrt{C_{x1}^2 + W_{\theta 1sh}^2}$	$378.3\frac{m}{s}$
1.15	Gas Constant	$R_{air} = \frac{(\gamma_a - 1)c_{pa}}{\gamma_a}$	$0.2871\frac{kJ}{mol \cdot K}$
1.16	Relative Mach Number at Tip	$M_{rel.sh} = \frac{W_{1sh}}{\sqrt{\gamma_a \cdot R_{air} \cdot T_{01}}}$	0.8767
1.17	Blade Velocity at Hub	$U_{1h} = r_{1h} \cdot w$	$316.1\frac{m}{s}$
1.18	Blade Velocity at Mean	$U_{1m} = r_{1m} \cdot w$	$358.2\frac{m}{s}$
1.19	Horizontal Relative Velocity at Hub	$W_{\theta 1h} = U_{1h} - C_{\theta 1}$	$255.8\frac{m}{s}$
1.20	Horizontal Relative Velocity at Mean	$W_{\theta 1m} = U_{1m} - C_{\theta 1}$	$298.0\frac{m}{s}$
1.21	Inlet Backsweep Angle at Hub	$\beta_{1h} = tan^{-1}(\frac{W_{\theta 1h}}{C_{x1}})$	$0.9965rad = 57°$
1.22	Inlet Backsweep Angle at Mean	$\beta_{1m} = tan^{-1}(\frac{W_{\theta 1m}}{C_{x1}})$	$1.064rad = 61°$
1.23	Relative Velocity at Hub	$W_{1h} = \sqrt{C_{x1}^2 + W_{\theta 1h}^2}$	$305\frac{m}{s}$
1.24	Relative Velocity at Mean	$W_{1m} = \sqrt{C_{x1}^2 + W_{\theta 1m}^2}$	$341\frac{m}{s}$
1.25	Relative Mach Number at Hub	$M_{rel.h} = \frac{W_{1h}}{\sqrt{\gamma_a \cdot R_{air} \cdot T_{01}}}$	0.7063
1.26	Relative Mach Number at Mean	$M_{rel.m} = \frac{W_{1m}}{\sqrt{\gamma_a \cdot R_{air} \cdot T_{01}}}$	0.7900
1.27	Mean Metal's Angle	$\beta^*_{1m} = \beta_{1m} - i_{1m}$	62°
1.28	Shroud Metal's Angle	$\beta^*_{1sh} = \beta_{1sh} - i_{1sh}$	57°
1.29	Hub Metal's Angle	$\beta^*_{1h} = \beta_{1h} - i_{1h}$	50°



(a) U_2 *Corrected for* $\beta_2^* = 0°$ [5]



(b) U_2 *Corrected for* $\beta_2^* = 35°$ [5]



(c) U_2 *Corrected for* $\beta_2^* = 45°$ [5]

Figure 2.4: Blade Velocity Corrected for Differing $\beta_2^\star$ Angles

Table 2.5: Stage 2 [5]

Step	Variable	Equation	Value
2.01	Exit Backsweep Angle	$0° \leq {\beta^*}_2 \leq 40°$	$31° = 0.5411 rad$
2.02	Flow Turning Angle	$\theta_1 = \frac{T_{01} \cdot 1.8}{518.7}$	1.607
2.03	Speed Ratio	Figure 2.4b	$1262 \frac{ft}{s}$
2.04	Impeller Exit Speed	$U_2 = (\frac{U_2}{\sqrt{\theta_1}}) \cdot \sqrt{\theta_1}$	$1599 \frac{ft}{s} = 487.5 \frac{m}{s}$
2.05	Outlet Temperature	$T_{02} = T_{03.partA}$	$662.8K$
2.06	Enthalpy's Change	$\Delta h_0 = c_{pa} \cdot (T_{02} - T_{01})$	$200.7 \frac{kJ}{kh}$
2.07	Horizontal Absolute Velocity	$C_{\theta 2} = \frac{\delta h_0 - C_{\theta 1} \cdot U_{1m}}{U_2}$	$367.4 \frac{m}{s}$
2.08	Exit Swirl	${\beta_2} > {\beta^*}_2$	$37° = 0.6458 rad$
2.09	Horizontal Relative Velocity	$W_{\theta 2} = U_2 - C_{\theta 2}$	$120.1 \frac{m}{s}$
2.10	Axial Velocity	$C_{r2} = \frac{W_{\theta 2}}{tan(\beta_2)}$	$159.4 \frac{m}{s}$
2.11	Relative Velocity	$W_2 = \frac{W_{\theta 2}}{tan(\beta_2)}$	$199.6 \frac{m}{s}$
2.12	Outlet Swirl	$\alpha_2 = tan^{-1}(\frac{C_{\theta 2}}{C_{r2}})$	$1.161 rad = 67°$
2.13	Absolute Velocity	$C_2 = \sqrt{C_{\theta 2}^2 + C_{r2}^2}$	$400.5 \frac{m}{s}$
2.14	Relative Velocity Ratio	$\frac{W_2}{W_{1sh}}$	0.5276
2.15	Meridional Velocity Ratio	$\frac{C_{r2}}{C_{x1}}$	0.9629
2.16	Thermo Work	$W_{thermo} = \frac{W_{HPC}}{5.216}$	$200.7 \frac{kJ}{kg}$
2.17	Slip Factor	$\sigma = 1 - \frac{0.63\pi}{N_{b3}}$	0.9381
2.18	Exit Backsweep Angle VERIFICATION	${\beta^*}_2 = tan^{-1}(\frac{U_2 - \frac{C_{\theta 2}}{\sigma}}{C_{r2}})$	$0.5416 rad = 31°$
2.19	Mass Flow Rate	$\dot{m}_2 = \dot{m}_1$	$5.216 \frac{kg}{s}$
2.20	Impeller Outlet Area	$A_2 = \frac{A_{1sh} \cdot C_{x1}}{C_{r2}}$	$0.01109 m^2$
2.21	B*aero	$0.12 \leq B^*_{2(aero)} \leq 0.17$	0.17
2.22	B*blade	$0.03 \leq B^*_{2(blade)} \leq 0.06$	0.05
2.23	B*2	$B^*_2 = B^*_{2(aero)} + B^*_{2(blade)}$	0.2150
2.24	Density	$\rho_2 = \frac{\dot{m}_2}{A_2 \cdot C_{r2}}$	$2.950 \frac{kg}{m^3}$

As depicted in Figure 2.5, the flow passing through the vaneless diffuser exhibits distinct stable and stalling regions, delineated by a blue curve. Designs positioned above this curve are considered stable. Our design, as shown in Figure 2.5, resides comfortably within the stable region. Therefore, the assumptions guiding its development were deemed valid.

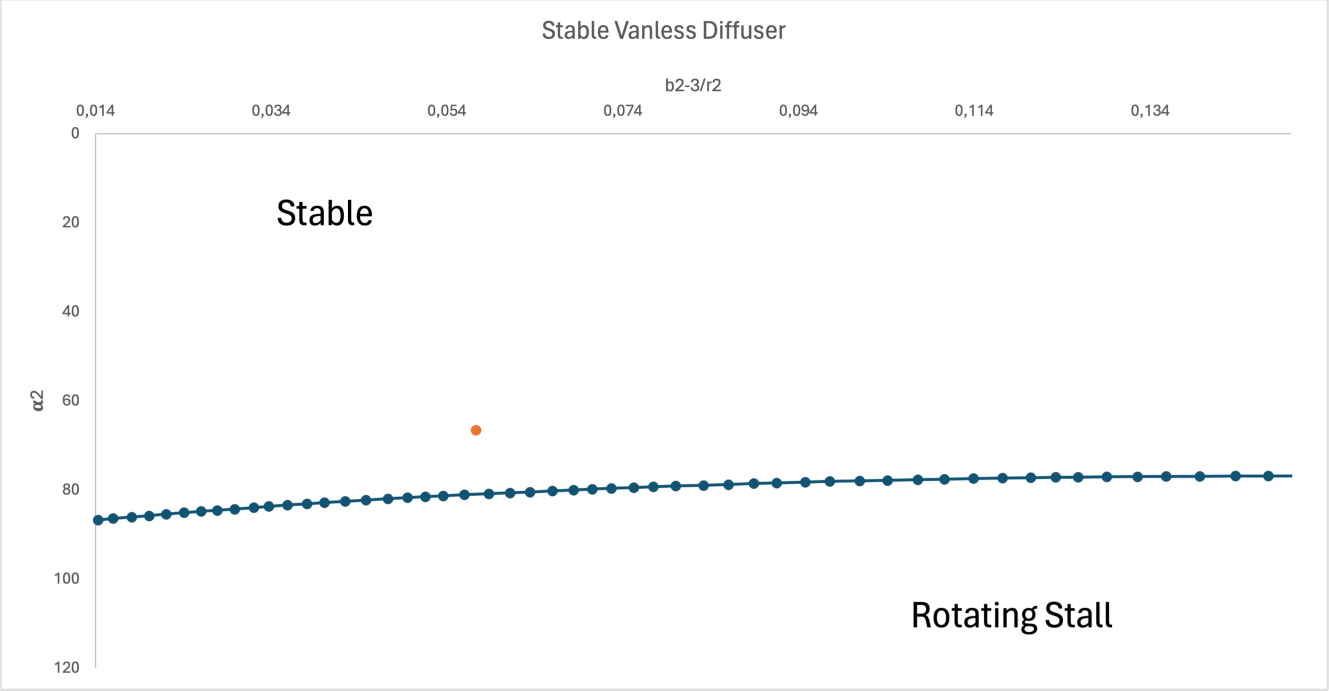


Figure 2.5: Vaneless Diffuser Stability [5]

2.3 Diffuser Design Parameters

The diffuser exit's design parameter values and equations are displayed in Table 2.6. The density was an important parameter to iterate in order to obtain accurate results at the throat. Finally, the throat aspect ratio is within the range of 0.8 and 1.2.

Table 2.6: Stage 3 [5]

Step	Variable	Equation	Value
3.01	Inlet Impeller Pressure	$P_{01} = P_{02:PartA}$	401.2 kPa
3.02	Adiabatic Efficiency	η was given	0.8850
3.03	Outlet Impeller Pressure	$P_{02} = P_{01}\left(\frac{\eta}{T_{01}}(T_{02}-T_{01})+1\right)^{\frac{\gamma_a}{\gamma_a-1}}$	1244 kPa
3.04	Vaneless Diffuser Exit Pressure	$P_{03} = 0.99 * P_{02}$	1232 kPa
3.05	Horizontal Absolute Velocity	$C_{\theta 3} = C_{\theta 2}\frac{r_2}{r_3}$	334.0 $\frac{m}{s}$
3.06	Diffuser Exit Area	$A_3 = 2\pi r_3 b_3$	6.186E-03 m^2
3.07	Density	ρ_3 was iterated	3.089 $\frac{kg}{m^3}$
3.08	Axial Velocity	$C_{r3} = C_2 * \left(\frac{r_2 * \rho_2}{r_3 * \rho_3}\right)$	347.8 $\frac{m}{s}$
3.09	Outlet Swirl	$\alpha_3 = tan^{-1}\left(\frac{C_{r3}}{C_{\theta 3}}\right)$	0.8055 rad $(46.15°)$
3.10	Absolute Velocity	$C_3 = \sqrt{C_{r3}^2 + C_{\theta 3}^2}$	482.2 $\frac{m}{s}$
3.11	Stagnation Outlet Temperature	$T_{03} = T_{03.partA}$	662.8 K
3.12	Outlet Mach number	$\sqrt{\frac{C_3^2}{T_{03}\left(\gamma_a R_{air} - \frac{1}{2}(\gamma_a - 1)\frac{C_3^2}{T_{03}}\right)}}$	1.028
3.13	Static Outlet Temperature	$T_3 = T_{03} - \frac{C_{r3}^2}{2c_{pa}}$	602.6 K
3.14	Static Pressure	$P_3 = P_{03}\left(\frac{T_{03}}{T_3}\right)^{\frac{\gamma_a - 1}{\gamma_a}}$	1265.5 kPa
3.15	Density VERIFICATION	$\rho_3 = \frac{\rho_1\left((\gamma_a+1)M_3^2\right)}{(\gamma_a-1)M_3^2+2}$	3.090 $\frac{kg}{m^3}$
3.16	Throat Blockage	B* was assumed	2.000E-02
3.17	Diameter	$d_3 = 2r_3$	0.2750 m
3.18	Flow Incidence Angle	i_3 is assumed	-0.05236 rad $(-3.000°)$
3.19	Inlet Vane Angle	$\alpha_3 * = \alpha_3 - i_3$	0.8579 rad $(49.15°)$
3.20	Diffuser Vane Count	$18 \leq N_v \leq 25$	18
3.21	Throat Temperature	$T_0^* = T_{03}$	662.8 K
3.22	Throat Pressure	$P_0^* = P_{03}$	1231.6 kPa
3.23	Throat Mass Flow Rate	$\dot{m}^* = \rho^* * A_3 * C_{r3} * 1.0125$	6.728 $\frac{kg}{s}$
3.24	Throat Mach Number	M^*	1.0
3.25	Throat Area	$\frac{1}{A^*} = \frac{\sqrt{g\gamma_a}(1-B^*)P_0^* M^*}{\dot{m}^*\sqrt{R_{air}T_0^*}}(1+(\frac{\gamma_a-1}{2})M^{*2})^{\frac{-1-\gamma_a}{2\gamma_a-2}}$	0.00113 m^2
3.26	Individual Throat Area	$a^* = \frac{A^*}{N_v}$	0.00006299 m^2
3.27	Diffuser Throat Width	$w^* = \frac{a^*}{b_3}$	0.0088 m
3.28	Throat Aspect Ratio	$AR_{th} = \frac{b^*}{w^*}$	0.8139

Section 3

High Pressure Turbine Design

This section covers the aerodynamic design and performance analysis of the high pressure axial turbine. The methodology discussed in the following subsections is presented in the chronological order of the design. Section 2 concludes with an overview of how the high pressure turbine code is implemented in the Python programming language.

Note that γ is equal to 1.333, c_{pg} is equal to 1.148 kJ/kgK and R is equal to 0.287 kJ/kgK in this section.

3.1 High Pressure Turbine Design Approach

An iterative approach has been used during the preliminary design phase of the high pressure turbine. Figure 3.1 presents a systematic approach taken by the turbine team to converge to a feasible design. The figure presents different systems used to navigate the feasible design space.

Several constraints have been strictly implemented throughout the design process to contain the results in a finite and physically valid design space. These design constrants and checks have been discussed in detail in the following subsections. The design space is unimodel and hence, a global search strategy has been adopted for optimization [2]. An XDSM diagram (Figure 3.1) visually represents the interactions and data dependencies between different components, aiding in understanding the complex system [2].

The following procedure is used to formulate the optimization problem in order to derive at a design.

1. Defining design variables.
    - Design Efficiency
    - Off Design Efficiency
    - Number of Rotors
    - Blade life
2. The method of weighted sums is used to define the optimization objective function. The idea is to combine all of the objectives into one objective function using weighted sums. However, the data

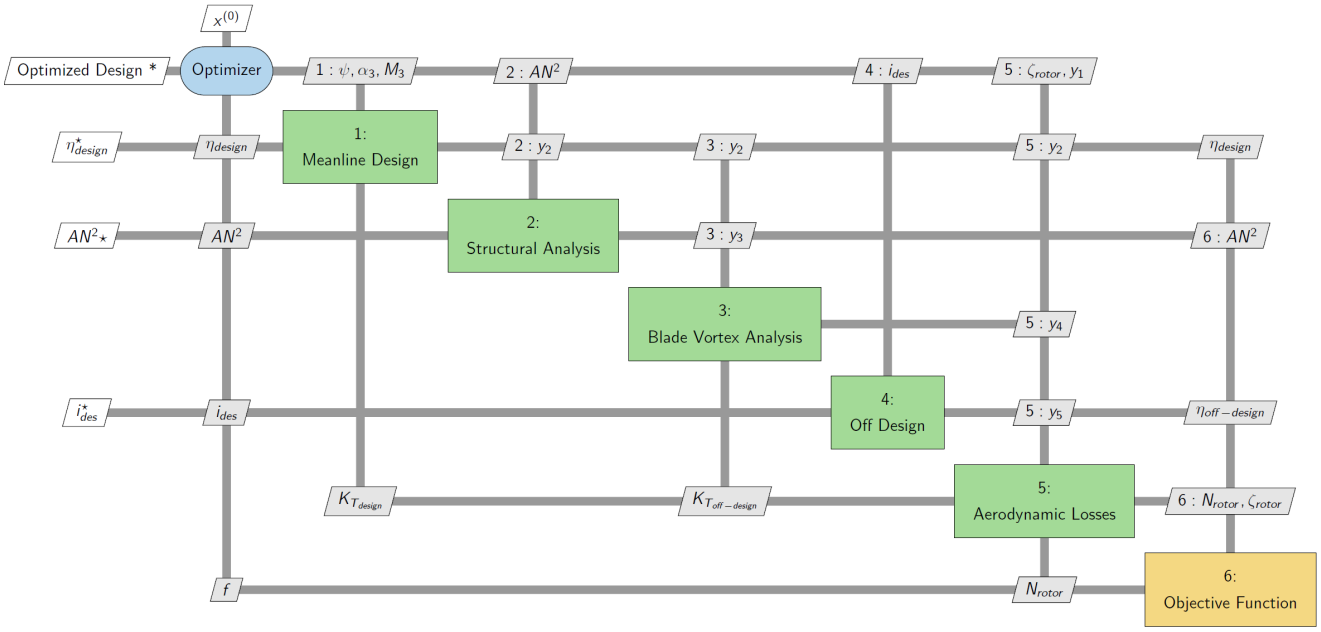


Figure 3.1: XDSM for High Pressure Turbine Design

is needed to be normalized first, thisis done using the following relation:

$$X_{\text{normalized}} = \frac{X - X_{\text{minimum}}}{X_{\text{maximum}} - X_{\text{minimum}}} \tag{3.1}$$

3. The optimization function looks as following. The method of weighted sums is used to obtain the result of the optimization function. The data is analyzed using a global search optimization strategy.

$$y = f(\eta_{\text{final}},\ \eta_{\text{final off-design}},\ N_{\text{rotor}},\ AN^2) \tag{3.2}$$

$$y = A\left(\frac{1}{\eta_{\text{final}}}\right) + B(\eta_{\text{final}} - \eta_{\text{final off-design}}) + C(N_{\text{rotor}}) + D(AN^2) \tag{3.3}$$

The objective function $y = f(x)$ is minimized to obtain the optimized based on the chosen weights. Where A, B, C and D are the weights that can be tuned to optimize the design.

3.2 Meanline Design

The meanline design covered in this section is the final result of the optimization process discussed previously. Note that from this section onward, station "1" (shown as a subscript 01 or 1) refers to the vane/nozzle inlet, while station "2" (subscript 02 or 2) is the vane exit/rotor inlet and station "3" (subscript 03 or 3) refers to the vane exit. Finally, angles denoted as α and β refer to the absolute and relative flow angles, respectively (the angle nomenclature used is consistent with Saravanamuttoo)

The table below summarizes several key input parameters. The reaction is temperature-based, the stage loading follows the Saravanamuttoo definition (twice that of Moustapha) and the α_3, mach at exit

(M_3) and AN^2 are within the ranges specified in the problem statement - note that each of these factors are inputs to the design process and the final values shown in the table below are the result of the optimization process described in the previous subsection. The shaft speed is also within the acceptable range provided by the compressor group. The temperature-based reaction of 0.527 is slightly higher than some turbines (typically in the 0.45-0.49 range [4]) because in this particular design, there is vane cooling assumed to be directly on the vanes while only disc cooling on the rotor. It is assumed that biasing the turbine slightly is acceptable under these conditions.

Table 3.1: Dimensionless Parameters and Process Inputs

Property	**Value**	**Source/Details**
R_{mean}	0.527	Temperature-based. Detailed in "Station 2" subsection below.
ψ	3.21 (or 1.605 Moustapha)	Within the standard range in both Moustapha and Saravanamuttoo. 3.21 selected on the basis of optimization, discussed later in this report.
U_{mean}	363.2 m/s	Computed directly from the stage loading using the work in Part A.
AN^2	$1.819x10^7 m^2 rpm^2$	Within the limits provided in the outline (units converted from imperial to metric). Selected based on optimization
ω	4214.5 rad/s	Within operating limits of the compressor. Selected based on optimization procedure.

The code example below shows how the stage loading is used to compute U_{mean}.

```
def calc_U(psi):
    """
    This function calculates the metal speed U.
    Input:  Stage loading coefficient "psi"
    Output: Returns the value of U
    """
    U = numpy.sqrt((2*c_p_gas*1000*(T_02_cycle - T_03)) / (psi))
    return U
```

Figure 3.2: Code snippet for calculation $U_{meanline}$.

Furthermore, the implementation of the meanline design in Python uses multiple checks to ensure that the design is conforming to physics and is following good design practices [1]. The table below summarizes the key checks used throughout the meanline design process.

Table 3.2: Physics and Design Checks

Property	Details
$V_3 > V_2$	Ensures that relative velocity increases across the rotor
$C_2 > C_1$	Ensures the absolute velocity increases across the stator
$U_{hub} < 335.28 m/s$	Ensures the limit of 1100 ft/s at hub is not exceeded
$\rho_1 > \rho_2 > \rho_3$	Ensures that density decreases across the turbine
$P_{03,rel} < P_{02,rel}$	Physics check ensuring that relative stagnation pressure decreases across the turbine
$M_{2,hub} < 1$	Ensures the largest value of absolute speed is below Mach 1
$M_{3,rel,tip} < 1$	Ensures the largest value of relative speed is below Mach 1

Finally, some additional key assumptions include:

- Increasing axial velocity across the turbine.
- The stator area is a converging duct
- The rotor area is constant ($A_2 = A_3$)

3.2.1 Station 1 - Nozzle/Vane Inlet

From cycle calculations and the given information in the problem statement, station 1 (vane inlet) is completely defined. The table below summarizes the key properties of station 1. The third column shows the source of the information (either "Cycle Calculations" from Part A, "Given in Problem Statement" or "Calculated") and detailed calculations follow the table for properties denoted as "Calculated".

Table 3.3: Station 1 - Vane/Nozzle Inlet

Property	Value	Source
T_{01}	1245.3 K	Cycle Calculations
P_{01}	1182.1 kPa	Cycle Calculations
$\dot{m}_1$	4.836 kg/s	Cycle Calculations
T_1	1242.1 K	Calculated
P_1	1169.8 kPa	Calculated
ρ_1	3.282 kg/m^3	Calculated
M_1	0.125	Given in Problem Statement
C_1	86.2 m/s	Calculated
C_{w1}	15 m/s	Calculated
C_{a1}	84.9 m/s	Calculated
α_1	-10 degrees	Given in Problem Statement
A_1	0.0174	Calculated

From the properties above, the meanline velocity triangle at station 1 is completed, as shown in the figure below (angles in degrees and velocities in m/s).

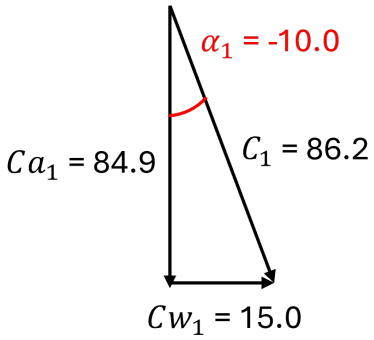


Figure 3.3: Velocity Triangle 1 - Nozzle Inlet

The code example below shows how the static temperature and pressure are obtained using the Mach number and the compressible flow equations. The static temperature is used to then compute the total speed (C_1), which with the given angle of α_1 = -10, is used to obtain C_{w1} and C_{a1}.

```
def calc_properties(M, T_stagnation, P_stagnation):
    T = T_stagnation/(1 + ((gamma_g - 1)/2) * M**2)
    P = P_stagnation/((1 + ((gamma_g - 1)/2) * M**2))**(gamma_g/(gamma_g - 1))
    rho = P/(0.287*T)
    c = M * numpy.sqrt(gamma_g * 287 * T)
    return T, P, rho, c
```

Figure 3.4: Code snippet for calculating gas properties.

The following subsection covers the mean line design at station 3 (rotor exit) which is necessary to complete before moving on to stage 2 (nozzle exit).

3.2.2 Station 3 - Rotor Exit

The table below summarizes the gas properties at station 3 (rotor exit) in addition to key parameters such the velocities and angles. As in the previous subsection, the third column shows the source of the information. Detailed explanations follow the table for properties denoted as "Calculated".

Table 3.4: Station 3 Properties - Rotor Exit

Property	Value	Source
T_{03}	1041.3 K	Cycle Calculations
P_{03}	517.9 kPa	Cycle Calculations
T_3	996.4 K	Calculated - Same approach as station 1 in previous subsection
P_3	433.6 KPa	Calculated - Same approach as station 1 in previous subsection
ρ_3	1.53 kg/m^3	Calculated - Same approach as station 1 in previous subsection
$\dot{m}_3$	5.06 kg/s	Cycle Calculations.
C_3	321.3 m/s	Calculated using Mach number and alpha 3 as inputs
C_{a3}	295.0 m/s	Calculated
C_{w3}	127.2 m/s	Calculated
U_{mean}	363.2 m/s	Calculated from Stage Loading
α_3	23.3 degrees	Range Given in Problem Statement. Optimization process resulted in this value being selected.
V_3	572.3 m/s	Calculated
V_{w3}	490.5 m/s	Calculated
ϕ_3	0.812	Calculated
β_3	59.0 degrees	Calculated
$M_{3,rel}$	0.93	Calculated
$P_{03,rel}$	743.3 kPa	Calculated
A_3	0.0123 m^2	Calculated

From the properties above, the meanline velocity triangles at station 3 can be completed are and displayed in the figure below (angles in degrees and velocities in m/s).

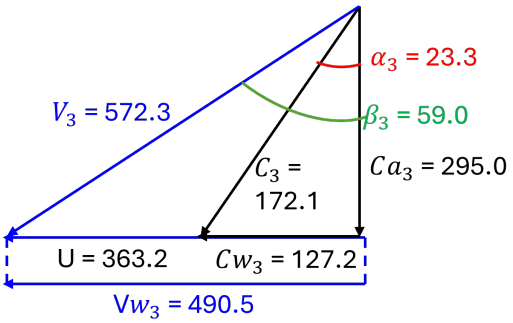


Figure 3.5: Velocity Triangle 3 - Rotor Exit

Similar to previous subsection, all properties listed as "Calculated" are detailed below. Note that the gas properties (ρ_3, P_3, T_3) and total absolute velocity C_3 are calculated using the exact same process as outlined in the previous subsection for station 1. Based on the approach taken by the team, Mach number at the exit and absolute exit angle are inputs within the range provided in the outline.

```
def calc_stage_3(U, C_3, T_3, rho_3,P_3,alpha_3):
    C_a_3 = C_3 * np.cos(np.radians(alpha_3))
    C_w_3 = math.sqrt(C_3**2 - C_a_3**2)
    V_w_3 = U + C_w_3
    alpha_3 = numpy.rad2deg(numpy.arcsin(C_w_3/C_3))
    V_3 = np.sqrt(V_w_3**2 + C_a_3**2)
    flow_coefficient_3 =  C_a_3 / U
    beta_3 = np.rad2deg(np.arctan(V_w_3 / C_a_3))
    a_3 = np.sqrt(gamma_g * R * 1000 * T_3)
    M_3_rel = V_3 / a_3
    A_3 = m_dot_3/(rho_3 * C_a_3)
    P_03_rel = P_3*(1+ (gamma_g-1)/2 * M_3_rel**2)**(gamma_g/(gamma_g-1))
    return C_a_3, C_w_3,  V_3, V_w_3, flow_coefficient_3, beta_3, a_3, M_3_rel, A_3, P_03_rel
```

Figure 3.6: Code snippet for calculating properties for turbine stage 3.

3.2.3 Station 2 - Nozzle Exit / Rotor Inlet

With station 3 completed, station 2 can now be calculated. The table below summarizes the gas properties at station 2 (nozzle exit) in addition to the key parameters such as the temperature-based reaction, velocities and angles.

Table 3.5: Station 2 Properties - Nozzle Exit / Rotor Inlet

Property	**Value**	**Source**
T_{02}	1225.9 K	Cycle Calculations
P_{02}	1182.2 kPa	Cycle Calculations
T_2	1122.3 K	Calculated
P_2	830.2 kPa	Calculated
ρ_2	2.577 kg/m^3	Calculated
$\dot{m}_2$	4.98 kg/s	Cycle Calculations
C_2	487.9 m/s	Calculated
C_{a2}	172.1 m/s	Calculated
C_{w2}	456.5 m/s	Calculated
U_{mean}	363.2 m/s	Calculated
α_2	69.3 degrees	Calculated
V_2	195.7 m/s	Calculated
V_{w2}	93.3 m/s	Calculated
ϕ_2	0.4737	Calculated
β_2	28.5 degrees	Calculated
$M_{2,abs}$	0.744	Calculated
$P_{02,rel}$	880.7 kPa	Calculated
A_2	0.0123 m^2	Equal to Area 3.

From the properties in the table above, the velocity triangle in the image below is derived for station 2. The velocities are in units m/s and the angles in degrees.

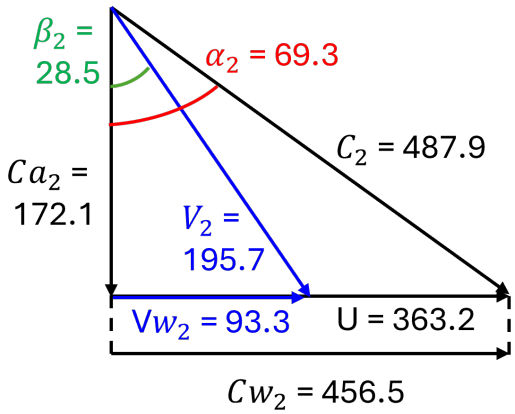


Figure 3.7: Velocity Triangle 2 - Rotor Inlet

Similar to previous sections, all properties listed as "Calculated" are detailed below. Note that this process requires iteration until convergence to ensure agreement between the equation of state (ideal gas law) and the conservation of mass. Also, V_{w2} and C_{w2}, together with previously determined V_{w3} and C_{w3}, ensure that the work always conforms to the turbine work needed in Part A. That is $Work = U_{mean}x(C_{w2}+C_{w3}) = U_{mean}x(V_{w2}+V_{w3})$ = 212 kJ/kg (as shown in Part A).Inputs that do not result in convergence are simply discarded and the next set of inputs is computed.

```
def calc_stage_2_trial(U,T_1,T_3,P_3,A_3,V_w_3):
    error = 1
    max_iterations = 10000
    count = 0
    T_2 = 1060
    increment = 0.01
    V_w_2 = (c_p_gas * 1000 * (T_02_cycle - T_03) / (U)) - V_w_3
    C_w_2 = (V_w_2 + U)

    while count < max_iterations:

        P_2 = P_01 * ((T_2/T_02_cycle) ** (gamma_g/(gamma_g - 1)))
        rho_2 = P_2/(R*T_2)
        T_02 = T_2 + (C_w_2**2 + (m_dot_2/(rho_2*A_3))**2)/(2*c_p_gas*1000)
        #print(T_02)
        error = np.abs(T_02_cycle - T_02)
        count = count + 1
        if (0 < error < 0.001):
            break
        else:
            T_2 = T_2 + increment

    if count != max_iterations:
        reaction = (T_2 - T_3)/(T_1 - T_3)
        A_2 = A_3
        C_a_2 = (m_dot_2)/(rho_2 * A_2)

        flow_coefficient_2 = C_a_2 / U
        a_2 = np.sqrt(gamma_g * R * 1000 * T_2)

        beta_2 = np.rad2deg(np.arctan(V_w_2 / C_a_2))
        V_2 = np.sqrt(V_w_2**2 + C_a_2**2)

        C_2 = np.sqrt(C_w_2**2 + C_a_2**2)
        alpha_2 = np.rad2deg(np.arctan(C_w_2/C_a_2))
        M_2 = C_2 / a_2
        M_2_rel = V_2 / a_2
        P_02 = P_2*(1+ (gamma_g-1)/2 * M_2**2)**(gamma_g/(gamma_g-1))
        P_02_rel = P_2*(1+ (gamma_g-1)/2 * M_2_rel**2)**(gamma_g/(gamma_g-1))

    return T_02, T_2, P_2, rho_2, A_2, C_a_2, flow_coefficient_2, a_2,
            V_w_2, beta_2, V_2, C_w_2, C_2, alpha_2, M_2, M_2_rel,P_02,P_02_rel, reaction

```

Figure 3.8: Code snippet for calculating properties for turbine stage 3.

3.3 Free Vortex Design

The free vortex design involves the calculation of the rotor hub and tip velocities and angles in addition to other parameters such as the blade height. It is assumed that the stagnation pressure and stagnation temperature are not a function of the radius, but that the static temperature and pressure do change with radius as the total speeds change radially.

Before calculating the hub and tip velocities and angles, the rotational speed, hub radius, tip radius, meanline radius and blade height need to be computed. The table below shows the final values of these properties.

Table 3.6: Blade Dimensional Parameters and Shaft Speed

Property	**Value**	**Source**
ω	4214.5 rad/s	Calculated
r_{hub}	0.0758 m	Calculated
r_{tip}	0.0966 m	Calculated
r_{mean}	0.0862 m	Calculated
h	0.0207 m	Calculated

The calculations below show how the values in the table above were established.

```
class aerostructural():
    def calc_structural(an_squared_pointer, area_2_pointer, U_meanline_pointer):
        N = numpy.sqrt((an_squared_pointer)/area_2_pointer)
        omega = N*2*numpy.pi/60
        r_meanline = U_meanline_pointer/omega
        h = (area_2_pointer * (N/60))/U_meanline_pointer
        r_hub = r_meanline - (h/2)
        r_tip = r_meanline + (h/2)
        return N, omega, r_hub, r_tip, r_meanline, h
```

Figure 3.9: Code snippet for calculating shaft speed and radial dimensions.

The hub triangles may now be computed based on the parameters above and the information from the preceding subsection. The table below shows the hub triangles, assuming a free vortex design.

Table 3.7: Hub Properties

Property	**Value**	**Source**
$T_{2,hub}$	1095.7 K	Calculated
$P_{2,hub}$	754.1 kPa	Calculated
$T_{3,hub}$	994.3 K	Calculated
$P_{3,hub}$	429.8 kPa	Calculated
$C_{2,hub}$	546.7 m/s	Calculated
C_{a2}	172.1 m/s	Calculated
$C_{w2,hub}$	520.4 m/s	Calculated
U_{hub}	319.5 m/s	Calculated
$C_{3,hub}$	328.5 m/s	Calculated
C_{a3}	295.0 m/s	Calculated
$C_{w3,hub}$	144.5 m/s	Calculated
$\alpha_{2,hub}$	71.7 degrees	Calculated
$\alpha_{3,hub}$	26.1 degrees	Calculated
$V_{2,hub}$	263.4 m/s	Calculated
$V_{w2,hub}$	199.4 m/s	Calculated
$V_{3,hub}$	550.5 K	Calculated
$V_{w3,hub}$	464.8 m/s	Calculated
$\beta_{2,hub}$	49.2 degrees	Calculated
$\beta_{3,hub}$	57.6 degrees	Calculated
$M_{2,rel,hub}$	0.41	Calculated
$M_{2,abs,hub}$	0.84	Calculated
R_{hub}	0.41	Calculated

The calculations below show how the values in the table above were established. Note that although not explicitly shown in the code below $C_{w2,hub}$, $C_{w3,hub}$, $V_{w2,hub}$ and $V_{w3,hub}$ are calculated from trigonometry using the angles in the table. The full code is available in the appendix.

```
def calc_hub_angles(r_m_pointer, r_hub_pointer, alpha_2_pointer, alpha_3_pointer,
flow_coeff_2_pointer, flow_coeff_3_pointer,
U_pointer, C_a_2, a_2,T_02, T_03,T_1,C_a_3):
        alpha_2_hub_rad = numpy.arctan((r_m_pointer/r_hub_pointer)
        *numpy.tan(numpy.deg2rad(alpha_2_pointer)))
        alpha_2_hub_deg = numpy.rad2deg(alpha_2_hub_rad)

        alpha_3_hub_rad = numpy.arctan((r_m_pointer/r_hub_pointer)
        *numpy.tan(numpy.deg2rad(alpha_3_pointer)))
        alpha_3_hub_deg = numpy.rad2deg(alpha_3_hub_rad)

        beta_2_hub_rad = numpy.arctan((r_m_pointer/r_hub_pointer)
        *numpy.tan(numpy.deg2rad(alpha_2_pointer)) - (r_hub_pointer/r_m_pointer)*
        (flow_coeff_2_pointer)**(-1))
        beta_2_hub_deg = numpy.rad2deg(beta_2_hub_rad)

        beta_3_hub_rad = numpy.arctan((r_m_pointer/r_hub_pointer)
        *numpy.tan(numpy.deg2rad(alpha_3_pointer)) + (r_hub_pointer/r_m_pointer)*
        (flow_coeff_3_pointer)**(-1))
        beta_3_hub_deg = numpy.rad2deg(beta_3_hub_rad)

        U_hub = U_pointer * (r_hub_pointer / r_m_pointer)

        V_2_hub = C_a_2/np.cos(beta_2_hub_rad)
        C_2_hub = C_a_2/np.cos(alpha_2_hub_rad)
        C_3_hub = C_a_3/np.cos(alpha_3_hub_rad)

        T_2_hub = T_02 - (C_2_hub**2)/(2*c_p_gas*1000)
        T_3_hub = T_03 - (C_3_hub**2)/(2*c_p_gas*1000)
        T_1_hub = T_1 #assumed since no free vortexing

        a_2_hub = np.sqrt(gamma_g*T_2_hub*R*1000)

        M_2_rel_hub = V_2_hub / a_2_hub
        M_2_hub = C_2_hub / a_2_hub

        reaction_hub = (T_2_hub-T_3_hub)/(T_1_hub-T_3_hub)

        return alpha_2_hub_deg, alpha_3_hub_deg, beta_2_hub_deg, beta_3_hub_deg,
        U_hub, V_2_hub, C_2_hub, M_2_rel_hub, M_2_hub,reaction_hub
```

Figure 3.10: Code snippet for calculating turbine hub properties.

Similar to the hub properties shown above, the table below contains the corresponding tip properties, assuming a free vortex design.

Table 3.8: Tip Properties

Property	Value	Source
$T_{2,tip}$	1140.7 K	Calculated
$P_{2,tip}$	886.0 kPa	Calculated
$T_{3,tip}$	997.8 K	Calculated
$P_{3,tip}$	436.6 kPa	Calculated
$C_{2,tip}$	442.3 m/s	Calculated
C_{a2}	172.1 m/s	Calculated
$C_{w2,tip}$	407.4 m/s	Calculated
U_{tip}	406.9 m/s	Calculated
$C_{3,tip}$	316.2 m/s	Calculated
C_{a3}	295.0 m/s	Calculated
$C_{w3,tip}$	113.8 m/s	Calculated
$\alpha_{2,tip}$	67.1 degrees	Calculated
$\alpha_{3,tip}$	21.1 degrees	Calculated
$V_{2,tip}$	172.1 m/s	Calculated
$V_{w2,tip}$	0.5 m/s	Calculated
$V_{3,tip}$	599.1 m/s	Calculated
$V_{w3,tip}$	521.4 m/s	Calculated
$\beta_{2,tip}$	0.18 degrees	Calculated
$\beta_{3,tip}$	60.5 degrees	Calculated
$M_{2,rel,tip}$	0.263	Calculated
$M_{2,abs,tip}$	0.675	Calculated
$M_{3,rel,tip}$	0.97	Calculated
R_{tip}	0.585	Calculated

The process in determining the tip values is exactly the same as outlined at the hub. For the sake of avoiding repetition, the code for the tip (identical process as the hub) is omitted from the body of the report but the full code is available in the Appendix.

3.4 Aerodynamic Losses

Turbine design optimization study must account for flow losses. Losses can be calculated using viscous, three-dimensional CFD analysis, but this approach has drawbacks due to its place in the design cycle [3]. Until precise blade shapes are known at a later stage, CFD analysis is not possible. Hence, a meanline analysis incorporating aerodynamic losses as functions of blade row inlet and exit velocity triangles, and overall geometric features is used for the scope of this analysis. The methodology is adapted from reference [3].

The profile loss is the result of skin friction on the surface and depends on the blade's contact area with the fluid, surface finish, Reynolds number, and Mach number of the flow. These effects are influenced by

the airfoil's geometry. Annulus losses are also caused by friction on the endwall surfaces [3].

Secondary flow, which consists of vortices resulting from boundary layers and passage curvature, causes some fluid to move in directions other than the main flow direction. Trailing vortices are formed at the blade's trailing edge when the flow separates, creating a wake. The wake's momentum deficit and the kinetic energy in the trailing vortices contribute to losses. Tip clearance losses occur in rotors when fluid leaks into the gap between the blade tip and the shroud, contributing little to no expansion work. The resulting tip leakage vortex interacts with corner and passage vortices, creating complex flow patterns within the passage.

Hence, the total pressure loss of a blade row is calculated as the sum of different losses as mentioned previously in this section. Equation 3.4 is the relation used.

$$K_T = K_p f_{Re} + K_s + K_{TE} + K_{clr} \tag{3.4}$$

3.4.1 Profile Loss

Profile loss coefficient is calculated based on a set of cascade set results. The profile loss coefficient is calculated as follows:

$$K_p = 0.914 \left(\frac{2}{3} K_p^{\star} K_{accel} + K_{sh} \right) \tag{3.5}$$

$$K_p^{\star} = \left\{ K_{P(\beta_1=0)} \left| \frac{\beta_1}{\alpha_2} \right| \left(\frac{\beta_1}{\alpha_2} \right) \left[K_{P(\beta_1=\alpha_2)} - K_{P(\beta_1=0)} \right] \right\} \left(\frac{t_{max}/c}{0.2} \right)^{(\beta_1/\alpha_2)} \tag{3.6}$$

Equation 3.6 can be used to interpolate for any given angle β_1 and α_2 using the graphs provided in reference [3]. To make the design process faster and to calculate the values for multiple design choices, the graphs were digitized and added to a database, where a linear regression machine learning model is used to provide the value for any given input.

Figure 3.11 presents a sample code for predicting the loss coefficients based on the provided. Since the loss calculation model discussed in Chapter 2 of reference [3] uses a lot of correlations which have been presented in forms of graphs, the same technique is used to predict the required values and will not be discussed again in the following sections.

```python
    def figure_2_3a(pitch_chord_ratio, exit_flow_angle):
        fig_2_3a = pd.read_csv(r'_input_database\figure_2_3a.csv')
        X = fig_2_3a[['pitch_chord_ratio', 'exit_flow_angle']]
        y = fig_2_3a['K_P_1']

        X, X_test, y, y_test = train_test_split(X, y, test_size=0.2)

        model = LinearRegression()
        model.fit(X_train, y_train)
        K_P_1 = model.predict(X)

        return K_P_1[0]
```

Figure 3.11: Example code for calculating profile loss coefficient, $K_{P,1}$.

$$K_{sh} = \left(\frac{\Delta p_0}{q_1}\right)_{sh} \left(\frac{p_1}{p_2}\right) \frac{1 - \left(1 + \frac{\gamma-1}{2} M_1^2\right)^{\gamma/(\gamma-1)}}{1 - \left(1 + \frac{\gamma-1}{2} M_2^2\right)^{\gamma/(\gamma-1)}} \tag{3.7}$$

where,

$$\left(\frac{\Delta p_0}{q_1}\right)_h = 0.75(M_{1,hub} - 0.4)^{1.75}$$

and

$$\left(\frac{\Delta p_0}{q_1}\right)_{sh} = \left(\frac{r_h}{r_t}\right) \left(\frac{\Delta p_0}{q_1}\right)_h$$

The effect of exit Mach number and channel flow acceleration is corrected for using the following relations:

$$K_{accel} = 1 - K_2(1 - K_1)$$

$$K_1 = \begin{cases} 1.0 & \text{for } M_2 \leq 0.2 \\ 1 - 1.25\,(M_2 - 0.2) & \text{for } M_2 > 0.2 \end{cases} \tag{3.8}$$

$$K_2 = (M_1/M_2)^2$$

3.4.2 Secondary Loss

Secondary losses experienced on the vane and blades are calculated next using the procedure outlined in the reference [3]. Because no graphs or figures were needed to determine the secondary loss coefficient variables, it could be simply calculated using the iterated outputs from the main Python code. The

following Equation 3.9 was used to find the secondary loss coefficient:

$$K_s = 1.2 \times K_s^* \times K_s \tag{3.9}$$

The 1.2 value figuring in the previous expression is used to compensate for the trailing edge losses that are calculated as a loss itself, and not integrated within profile or secondary loss calculations. Furthermore, K_s is known as the subsonic Mach number correction factor for secondary losses, which is added to adjust the loss predictions for the effects of compressibility at subsonic speeds. Finally, K_s^* is the uncorrected secondary loss coefficient. The latter is found by using Equation 3.10:

$$K_s^* = 0.0334 f_{(AR)} \left(\frac{\cos\alpha_2}{\cos\beta_1}\right) \left(\frac{C_L}{s/c}\right)^2 \frac{\cos^2\alpha_2}{\cos^3\alpha_m} \tag{3.10}$$

Where $f_{(AR)}$, known as the aspect ratio function, is found using Equation 3.11:

$$f_{(AR)} = \begin{cases} \frac{1-0.25\sqrt{2-h/c}}{h/c} & \text{for } h/c \leq 2 \\ \frac{1}{h/c} & \text{for } h/c > 2 \end{cases} \tag{3.11}$$

Where $\frac{C_L}{s/c}$, being the lift coefficient to pitch-to-chord ratio, is found with Equation 3.12:

$$\frac{C_L}{s/c} = 2\left(\tan\alpha_1 + \tan\alpha_2\right)\cos\alpha_m \tag{3.12}$$

Finally, α_m, is known as the mean gas angle and is calculated with Equation 3.13:

$$\alpha_m = \tan^{-1}\left[\frac{1}{2}\left(\tan\alpha_1 - \tan\alpha_2\right)\right] \tag{3.13}$$

α_1 and α_2 are the relative and absolute gas angles for the vane and blade. The last element required to find the secondary loss coefficient, K_s, can be found using the following Equation 3.14:

$$K_s = 1 - K_3(1 - K_{accl}) \tag{3.14}$$

K_{accl} is the same variable as the one used for profile loss subsonic Mach number correction, while K_3 is simply the inverse of the aspect ratio squared.

Once all the necessary coefficients were obtained, the secondary loss coefficients that were found were verified for both the stator and the rotor against typical literature secondary loss coefficient values. These values placed the secondary loss coefficients from this design at a very low positive incidence angle (below 10°) in the typical breakdown of losses chart figure 2.2 of reference [3].

3.4.3 Trailing Edge Loss

Because this design does not use the basic types of blades, such as impulse blading and axial entry nozzle, the coefficient required from figure 2.10 of reference [3], in order to calculate trailing edge losses has to be interpolated. This is done by using the trailing edge thickness to throat opening ratio and applying

it to both basic types of blades within the figure. Lastly, the values are then inserted into the following Equation 3.15, to find $\Delta\phi^2_{\text{TE}}$, an energy coefficient:

$$\Delta\phi^2_{\text{TE}} = \Delta\phi^2_{\text{TE},\beta_1=0} + \left|\frac{\beta_1}{\alpha_2}\right| \left(\frac{\beta_1}{\alpha_2}\right) \left[\Delta\phi^2_{\text{TE},\beta_1=\alpha_2} - \Delta\phi^2_{\text{TE},\beta_1=0}\right] \tag{3.15}$$

The energy coefficient is then used to find the trailing edge loss coefficient using Equation 3.16:

$$K_{\text{TE}} = \frac{1 - \frac{\gamma-1}{2}M_2^2 \left(\frac{1}{1-\Delta\phi^2_{\text{TE}}} - 1\right)^{-\frac{\gamma}{\gamma-1}} - 1}{\left[1 - \left(1 + \frac{\gamma-1}{2}M_2^2\right)^{-\frac{\gamma}{\gamma-1}}\right]} \tag{3.16}$$

γ is the gas constant, while M_2 is the Mach number at the exit of the stator or rotor, depending on which is being calculated.

Another important aspect of trailing edge losses is it allows to find the number of blades for the rotor and the number of vanes for the stator. This is a very important parameter of the design, as it will have a high influence on many aspects, such as performance, weight, noise, etc. This is possible using variables found and needed through trailing edge losses and is found using the following Equation 3.17:

$$N = \frac{2\pi \cdot r_{\text{meanline}}}{\text{Pitch}} \tag{3.17}$$

3.4.4 Tip Clearance Loss

One of the given design parameters is for an unshrouded rotor, which influences the efficiency drop due to tip clearance losses. This can be found using figure 2.21 from reference [3]. Using the tip clearance to blade span ratio, it can be iterated for an approximate efficiency change. However, for a more detailed and precise analysis, the first step is to find the turbine efficiency, for which the following Equation 3.18 is used:

$$\eta_{tt} = \frac{1}{\left\{1 + \left[\frac{(\zeta_N * V_2^2 + \zeta_R * V_{r3}^2)}{2\,(h_{01} - h_{03})}\right]\right\}} \tag{3.18}$$

The turbine rotor (ζ_R) and nozzle (ζ_N) loss coefficient values for the latter, are found using the Equation 3.19:

$$\zeta_R = \frac{K_T}{1 + 0.5\gamma M_{r3}^2} \tag{3.19}$$

Once the overall turbine efficiency at 0 tip clearance is calculated, the unshrouded Equation 3.20 for efficiency loss due to tip leakage is used:

$$\frac{\left(\frac{\Delta\eta}{\eta_0}\right)}{\left(\frac{\Delta t_c}{h\cos\alpha_2}\right)\frac{r_t}{r_m}} = 0.93 \tag{3.20}$$

This equation gives the efficiency drop due to losses including tip clearance losses, which becomes the overall efficiency of the designed HPT.

3.4.5 Aerodynamic Losses Results

The following section presents the results computed using the methodology in the previous section.

Table 3.9: Loss Calculation Results.

Stator Losses		**Rotor Losses**	
$K_{p,stator}$	0.027602	$K_{p,rotor}$	0.029181
$K_{s,stator}$	0.097244	$K_{s,rotor}$	0.079922
$K_{TET,stator}$	0.009733	$K_{TET,rotor}$	0.021612
K_{stator}	0.134580	K_{rotor}	0.130715

Table 3.10: Stator and Rotor Geometry Values - meanline.

Stator Geometry		**Rotor Geometry**	
$c_{true,stator}$	0.052807 m	$c_{true,rotor}$	0.015955 m
$c_{axial,stator}$	0.029256 m	$c_{axial,rotor}$	0.014141 m
Φ_{stator}	56.3568^o	Φ_{rotor}	27.5833^o
$Pitch_{stator}$	0.040356	$Pitch_{rotor}$	0.010865
o_{stator}	0.014234 m	o_{rotor}	0.005600 m
N_{stator}	13	N_{rotor}	49
ζ_{stator}	0.877338	ζ_{rotor}	0.915409

Figure 3.12 shows the geometric parameters of the final rotor blade design.

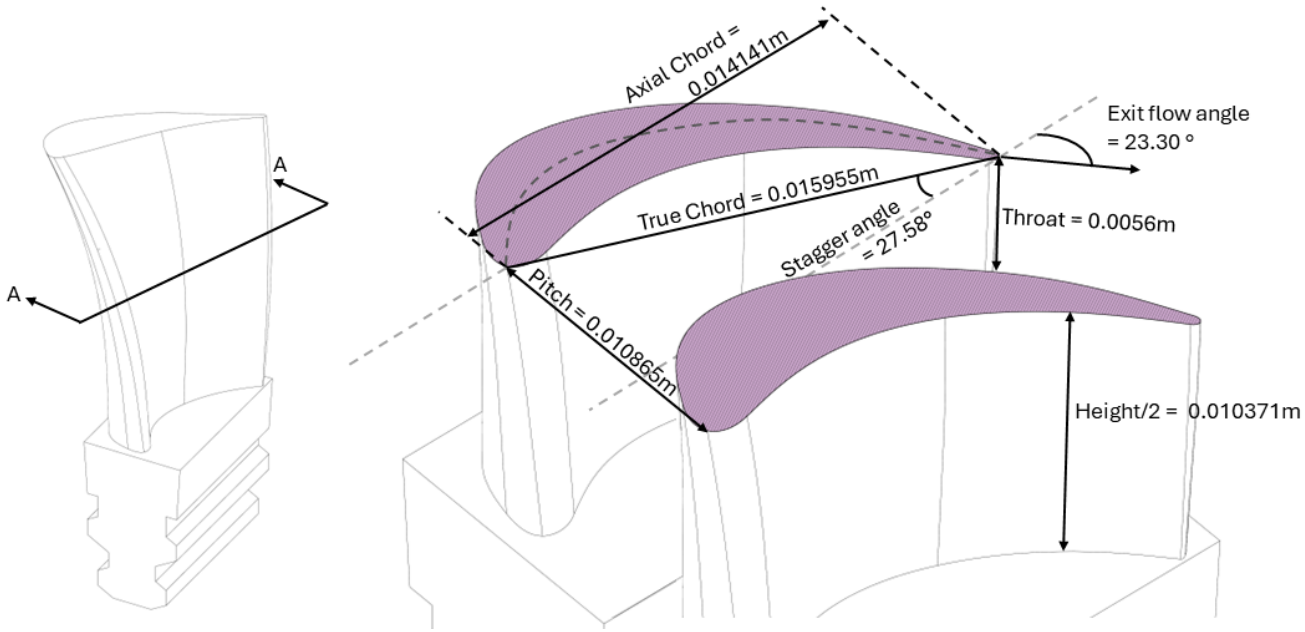


Figure 3.12: Detail Blade Geometry

3.5 HPT Off-Design

3.5.1 HPT Off-Design Meanline Analysis

The off-design analysis begins, in which the blade speed is reduced by 10 percent (from 363.2 m/s to 326.9 m/s at the meanline), begins with the following assumptions:

- Assuming the mass flow rate is constant between on-design and off-design.
- Assuming the vane exit properties are the same as in design conditions
- Assume blade exit relative flow angle is the same as in design conditions (deviation = 0)

The table below summarizes the key off-design parameters.

Table 3.11: Off-design Velocities and Angles

Stage 2		**Stage 3**	
α_2	69.35°	α_3	29.91°
β_2	36.99°	β_3	58.98°
C_{a_2}	172.06	C_{a_3}	300.6
V_{w_2}	129.61	V_{w_3}	499.79
V_2	215.41	V_3	583.22
C_2	487.86	C_3	346.77
C_{w_2}	456.51	C_{w_3}	172.89
U_{OD}	326.9	U_{OD}	326.9

The image below shows the off-design velocity triangles compared with the on-design triangles. Note that the orange-dotted triangles are the design point triangles.

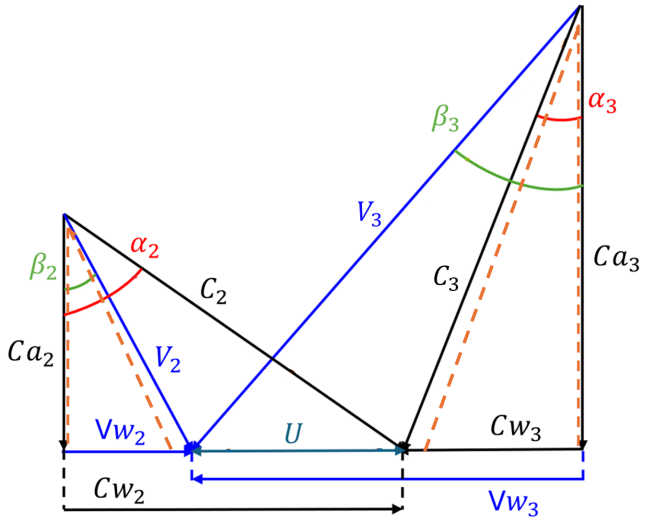


Figure 3.13: Off-design and On-design velocity triangles

In addition, the off-design pressures, temperatures, work and incidence are shown in the table below for the exit.

Table 3.12: Off-design gas properties and incidence

$T_{3,od}$	988.9 K
$T_{03,od}$	1041.3 K
$P_{3,od}$	421.32 kPa
$\rho_{3,od}$	1.48 kg/m^3
W_{od}	205.7 kJ/kg
$M_{3,rel,od}$	0.948
$Incidence$	8.52 degrees

The Python code below shows how the off-design parameters in the tables above were computed. In short, Ca_3 is iterated until the final Ca_3 satisfies both the equation of state (ideal gas law) and the conservation of mass for the given conditions and known exit geometry. The "LHS" variable is derived from beginning with the ideal gas law, then substituting ρ for P/RT. The LHS equation becomes the known constants of $\dot{m}$, the gas constant R and known exit area. The RHS of the equation is equal to $Ca_3 * P/T$ which are all ultimately functions of Ca_3 as shown in the code. Therefore, by iterating Ca_3 until the RHS and LHS of the equations are in agreement (within some tolerance), then both the equation of state and conservation of mass are satisfied.

```
class off_design():
    def calc_off_design(A_3, U_mean,beta_3,Ca_2, Cw_2, beta_2, a_2, a_3):
        U_mean_od = U_mean *0.9
        C_w_2_od = Cw_2
        flow_coeff_2_od = Ca_2/U_mean_od
        V_w_2_od = Cw_2 - U_mean_od
        beta_2_od = np.arctan(V_w_2_od/Ca_2)
        beta_2_od_deg = np.rad2deg(alpha_2_rel_od)
        incidence_2 = alpha_2_rel_od_deg - beta_2
        v_2_od = np.sqrt(V_w_2_od**2 + Ca_2**2)
        M_2_rel_od = v_2_od / a_2
        T_2 = a_2**2/(gamma_g*R*1000)
        LHS = R*m_dot_3/A_3
        error_threshold = LHS * 0.01 #1 percent error of the LHS
        Ca_3_range = np.linspace(100,400,1000)
        Ca_3_od = 0
        for i in Ca_3_range:
            V_w_3_od = i * np.tan(np.deg2rad(beta_3))
            C_w_3_od = V_w_3_od - U_mean_od
            C_3_od = np.sqrt(i**2 + C_w_3_od**2)
            T_3_od = T_03 - (C_3_od**2)/(2*1000*c_p_gas)
            P_3_od = P_03 * (T_3_od/T_03)**(gamma_g/(gamma_g-1))
            RHS = i *P_3_od/T_3_od
            a_3_od = math.sqrt(gamma_g*R*1000*T_3_od)
            if np.abs(LHS - RHS) < error_threshold:
                Ca_3_od = i
                alpha_3_od = np.rad2deg(np.arctan(C_w_3_od/Ca_3_od))
                rho_3_od = P_3_od/(R*T_3_od)
                work_od_cw = U_mean_od*(C_w_3_od + C_w_2_od)
                work_od_vw = U_mean_od*(V_w_3_od + V_w_2_od)
                flow_coeff_3_od = Ca_3_od/U_mean_od
                V_3_od = np.sqrt(V_w_3_od**2 + Ca_3_od**2)
                M_3_rel_od = V_3_od/a_3_od
                # For physics check
                P_2 = P_3_od * ((T_2/T_3_od) ** (gamma_g/(gamma_g - 1)))
                P_02_rel = P_2*(1+ (gamma_g-1)/2 * M_2_rel_od**2)**(gamma_g/(gamma_g-1))
                P_03_rel = P_3_od*(1+ (gamma_g-1)/2 * M_3_rel_od**2)**(gamma_g/(gamma_g-1))
                break
            return T_3_od, rho_3_od, P_3_od, alpha_3_od, beta_2_od_deg, flow_coeff_2_od,
            incidence_2, v_2_od, C_w_3_od,Ca_3_od,U_mean_od,flow_coeff_3_od, work_od_cw,
            work_od_vw, M_2_rel_od, M_3_rel_od, P_02_rel, P_03_rel
```

Figure 3.14: Code snippet for calculating turbine hub properties.

The following subsection outlines the losses that result from the 8.52 degrees of incidence observed above.

3.5.2 HPT Off-design Losses

The same procedure as for the meanline design is used in terms of losses, such as finding the profile, secondary, trailing edge and tip clearance losses. Because trailing edge and tip clearance losses are calculated using the same equations as for on-design, only the profile and secondary losses calculations are shown. Throughout this section, snippets of code are attached to each loss to showcase and explain the design process.

For profile losses, Figure 3.15 shows the code used to find the profile losses for off-design. Firstly, the following Equation 3.21 was used to find the necessary correlation variables to find profile loss:

$$\phi_{P0}^2 = \frac{1}{1 + \frac{K_{P0}}{K_1 + K_2 K_{P0}}} \tag{3.21}$$

In the meantime, Figure 2.34 from reference [3] was used to get the 2nd constant needed for the profile loss correlation formula. In that graph, the x-axis is represented by this Equation 3.22:

$$\left(\frac{d}{s}\right)^{-1.6} \left(\frac{\cos\beta_{1b}}{\cos\beta_{2b}}\right)^{-2} \left(\beta_1 - \beta_{1,\text{des}}\right) \tag{3.22}$$

Once both values were found, ϕ-squared could now be calculated with Equation 3.23:

$$\phi_P^2 = \phi_{P0}^2 - \phi_{\text{INC}-P}^2 \tag{3.23}$$

Finally, the last step is to plug the latter into the off-design profile loss Equation 3.24:

$$K_p = \frac{K_1(1-\phi_p^2)}{\phi_p^2 - K_2(1-\phi_p^2)} \tag{3.24}$$

```
def off_design_k_p(inputs):
        phi_squared_P0 = 1 / (1 + ((K_p_rotor) / (K_1_rotor + K_2_rotor * K_p_rotor)))
        s = pitch_chord_ratio_rotor * c_true
        d_s = LE_diameter_rotor/s
        graph_x = (d_s)**(-1.6) * (np.cos(np.radians(beta_2)) /
        np.cos(np.radians(beta_3)))**(-2) * (np.radians(incidence))

        def figure_2_34(graph_x):
            fig_2_34 = pd.read_csv(r'_input_database\figure_2_34.csv')
            x = fig_2_34['graph_x'].values
            y = fig_2_34['deg_of_accel'].values

            interp_func = interp1d(x, y, kind='cubic')
            interpolated_y = interp_func(graph_x)

            return interpolated_y
        deg_of_accel = figure_2_34(graph_x)
        phi_squared_P = phi_squared_P0 - deg_of_accel
        K_p_od = (K_1_rotor * (1-phi_squared_P)) / (phi_squared_P - K_2_rotor *
        (1-phi_squared_P))
```

Figure 3.15: Code snippet for calculating off design primary losses

Afterwards, for the secondary losses, the process was simpler, where a ratio of the design and off-design secondary losses was obtained through the figure 2.35 from the reference [3]. Once that was done, the rest of the procedure was simply interpolating the various data points to optimize for the best value, which is outlined in the snippet of code of Figure 3.16.

```
def off_design_k_s(inputs):
        d_c = LE_diameter_rotor/c_true
        graph_x_2 = (d_c)**(-0.3) * (np.cos(np.radians(beta_2)) /
        np.cos(np.radians(beta_3)))**(-1.5) * ((np.radians(incidence))/(np.radians(beta_2)
        + np.radians(beta_3)))
        if graph_x_2 < 0.27:
            def figure_2_35(graph_x_2):
                fig_2_35 = pd.read_csv(r'_input_database\figure_2_35.csv')
                x = fig_2_35['x'].values
                y = fig_2_35['K_K_des'].values
                interp_func = interp1d(x, y, kind='cubic')
                interpolated_y = interp_func(graph_x_2)
                return interpolated_y

            K_K_des = figure_2_35(graph_x_2)
            K_s_od =  K_K_des * K_s_rotor
```

Figure 3.16: Code snippet for calculating off design secondary losses.

Afterwards, the trailing edge losses and tip clearance losses were calculated using the same method as on-design meanline aerodynamic losses, where trailing edge losses permitted to output all the various blade geometry features, and tip clearance allowed to calculate the final efficiency following the decrease from aerodynamic losses. Once those same calculations were done for the rest of the off-design losses, the new off-design total-to-total efficiency was calculated, showing how much the off-design was affected by it. The final off-design efficiency for this design holds at a 87.21%, following the addition of a -4 degree incidence. For all the efficiency values extracted from this process and outputted from the code, see Table 3.13

3.6 Efficiency Calculation

The following table, Table 3.13 compiles all the calculated efficiencies, including total-to-total and final efficiencies for both on design and off-design parameters:

Table 3.13: Calculated Design Efficiencies

Efficiency	**Value**	**Comments**
η_{tt}	90.26 %	Turbine total-to-total efficiency
η_{final}	88.50 %	Final design efficiency
$\eta_{final,od}$	86.96 %	Off design efficiency
$\eta_{final,od,new}$	87.21 %	Off design efficiency after adding design incidence
$\Delta\eta_{final}$	1.13 %	Off design efficiency drop

Section 4

Trade Studies

4.1 Trade Study 1: Blade Material

The selection of a turbine blade material is one which should be taken with great care. These materials must be able to withstand extremely high temperatures while also maintaining their structural integrity at extremely high rotational speeds. Due to these extremely harsh environments, chromium nickel alloys are often chosen. With their high strength and low density, along with its ease of manufacturing make it a favorite in the aerospace industry.

In this trade study, this alloy shall be used as a baseline material, and compared with materials that have similar thermal and stress characteristics, while having both higher and lower densities and varying lifespans. These differences will be used to calculate the impact on the cost to the consumer which shall be compared to the overall lifetime cost of an engine.

The materials used in the trade study will be shown as such: Material X shall be the baseline material. The baseline density of 0.296kg/m3 shall be used, along with characteristics such as melting point, yield stress, etc. This material has a Factory Standard Cost (FSC) of 150 USD and will be rated at 5000 hours of life capability. Material Y offers a cheaper solution, with an FSC of 125USD per blade, a 5 percent lower density, and a life capability of 4000 hours. Material Z is a more durable solution, offering an FSC of 175USD per blade, with a 5 percent higher material density, and a lifecycle capacity of 8000 hours. These values are also summarized in the table below.

Table 4.1: Summary of Trade Study Materials

Material	**FSC (USD)**	**Life Capability (hrs)**	**Density (to baseline)**
Material X	$150	5000	–
Material Y	$125	4000	-5%
Material Z	$175	8000	+5%

For each blade configuration, the number of overhauls required will also be calculated. The timing of an overhaul shall be determined by the turbine blade life capability outlined above. Each overhaul shall be estimated to cost the client 90,000USD. The number of total overhauls is assumed based on an estimated engine lifespan of 25,000 hours, with an average of 2,500 hours per year.

Further, an increase of 60 percent shall be added to the blade costs as a sale markup. With this information, the total cost to the client has been calculated and tabled below.

Table 4.2: Lifetime Cost per Engine

	Material X	**Material Y**	**Material Z**
Sale Price, Blades (USD)	9,300	7,750	10,850
Lifetime Overhaul Cost** (USD)	450,000	540,000	270,000
Overhauls Required	5	6	3
Total Cost (USD)	**464,880**	**552,400**	**287,360**

From the table above, although Material Z has a higher upfront cost, the lifetime cost savings are substantial. Further, with an overhaul taking time away from the working time of the aircraft, the operator's losses due to the aircraft remaining stationary for weeks cannot be ignored. By having two less scheduled overhauls, the client will not only save over 180,000 USD, but gain weeks of usable time for their aircraft. Unfortunately, the cost of choosing Material Z is an increase in weight of 5 percent. To help reduce this weight increase, another trade study was conducted to reduce the engine weight.

4.2 Trade Study 2: Weight Reduction

As with any element of an aircraft, the engine shall be designed with weight considerations in mind. As such, a trade study to reduce the engine weight has been conducted. To reduce the weight, an attempt at increasing the engine efficiency has been made. To increase the efficiency, the peak cycle temperature can be increased, resulting in a decrease in specific fuel consumption (SFC). While keeping the output power of the engine the same, a lower volume of fuel should be needed, leading to a reduction in weight. This reduction in fuel weight should also save the operator in running costs, as the engine becomes more fuel efficient.

To accommodate for the increase in temperature, additional cooling shall be provided to the HPT such that for every 100°F of increase, cooling increases 1 percent from baseline. This will in turn reduce the HPT efficiency by 0.2 percent. The primary reduction in weight would focus on the amount of fuel consumed by the engine, as with an increase in peak temperature causing an increase in efficiency, less fuel consumption will occur.

The increases in temperature were chosen to be ever 100°F up until approximately 2000°F as this was to be considered the design limit of the material, while considering the extra cooling. These initial points were split further to allow for more detailed analysis, adding four points between each temperature. This led to a total of 12 data points for consideration. Due to the large processing time associated with this computation, the solution was not refined further.

By inserting the various temperatures into the Python script, the engine SFC, and air-to-fuel ratio are output given the power output remains constant. This then allows for the fuel consumption to be calculated, which will then demonstrate the decrease in weight due to fuel. The tabulated data for the key points of increases of 100°F is shown below.

Table 4.3: Results of Increasing Peak Cycle Temperature

Peak temp (K)	**SFC**	**Fuel to air ratio**	**Fuel Operational Cost (USD/hr)**	**Fuel weight (lbs)**
1245	0.41761	0.2	402.52	751.38
1301	0.36369	0.017416	350.55	654.36
1357	0.32323	0.015478	311.56	581.57

From this analysis, a sharp decrease in the fuel consumption per hour can be observed. With the rising cost of fuel and companies continually being required to lower emissions, this could prove as a vital selling point of the engine. This in turn means that 169.81 lbs can be saved for every hour of flight, proving to also be economical in terms of the weight of the propulsion system.

Furthermore, we may also reduce weight through other means. If the client deems the weight loss outweighs the significant cost increase for overhauls, a lighter material such as Material Y could be chosen. This however is not recommended, as the gains would be too few to warrant the cost. Alternatively, the turbine can be optimized to use the least number of blades possible. This would reduce the sale cost to the client, as well as save weight. However, the blades would be more highly loaded, but would still be within the acceptable range to maintain the promised life capability.

The result of the trade studies has allowed for a more cost efficient and marketable engine. Based on the conducted trade studies, the final recommendation is as follows:

1. Select Material Z for the turbine blades: This material's long-life capability proves that it is the most cost-effective option and the weight gain from choosing this option can be mitigated.

2. Increase the peak cycle temperature to 1357K: Increasing the peak cycle temperature allows for a more fuel-efficient engine and saves weight on fuel.

3. Reduce the blade count in the turbine: Reducing the number of blades in the rotor and stator of the turbine will keep the sale cost low for the client, while saving weight on engine design.

Section 5

Conclusion

To conclude this technical report, the optimization process for the gas turbine engine design was presented with detailed sections for the cycle calculations, high pressure compressor, high pressure turbine, and finally the trade studies. In the cycle calculation the overall engine cycle was covered and afterwards, a detailed high pressure compressor design is shown, followed by an extensive high pressure turbine design. This included the meanline design, free vortex design, aerodynamic losses and finally the off-design parameters. Finally, the trade studies were presented in order to select an appropriate blade material and decrease the overall weight.

The turbine conforms to all constraints provided in the outline, and conforms to physics. The turbine design summarized in this paper is the result of an optimization process discussed previously and takes into account engine efficiency, efficiency drop during off-design operation, number of rotor blades and the structural limit of the blade (via AN^2).

Column 1	**Column 2**
η_{final}	88.50%
$\eta_{offdesign}$	87.21%
AN^2	1,819,183
$M_{abs,exit}$	0.522
α_{exit}	23.30 degrees
ω	4214.5 rad/s

Table 5.1: Turbine Characteristics

The compressor in turn functions within the given constraints, including a relative Mach number below 1, aspect ratio of the diffuser is between 0.8 and 1.2 and the relative (0.5 and 0.6) and axial velocity range (between 0.8 and 1).

By incorporating trade studies into the design process, a more refined, optimized solution was determined and this aims to reduce the cost and downtime of the engine to the client. By following the recommendations outlined in this report, the final design showcases an engine which proves to be competitive in an ever-growing market.

Bibliography

[1] Saeed Farokhi. *Aircraft propulsion : cleaner, leaner, and greener*. Wiley, Hoboken, NJ, third edition edition, 2022.

[2] Joaquim R. R. A. Martins and Andrew Ning. *Engineering Design Optimization*. Cambridge University Press, Cambridge, UK, January 2022.

[3] Hany Moustapha. *Axial and Radial Turbines*. Concepts NREC, White River Junction, Vt., 2003.

[4] H. I. H. Saravanamuttoo. *Gas Turbine Theory*. Pearson, Harlow, England, 7th edition, 2017.

[5] Tsukasa Yoshinaka. Centrifugal compressor design. pages x, 90, 2024.

APPENDICES

Space left blank on purpose.

Appendix A

Python Script

Space left blank on purpose.

In [ ]:
```
import math
import numpy as np
```

Define Engine Characteristics

In [ ]:
```
# Provided Engine Characteristics
mechanical_eff = 0.99
gamma = 1.4
gamma_g = 1.33333
c_p_air = 1.005
c_p_gas = 1.148
# ENGINE INLET
inlet_loss = 0.01           # Inlet pressure loss
# ------------------------------------------------------------------------------
# Compressor
m_dot = 5.21                # [lb/s]
lpc_pr = 4                  # LPC PR
lpc_eff = 0.865             # LPC Target Efficiency
hpc_pr = 3                  # HPC PR
hpc_eff = 0.855             # HPC Target Efficiency
hpc_bleed_air = 0.09        # HPC Bleed Air (exit)
# ------------------------------------------------------------------------------
# Combustor
AFR = 0.02                  # Fuel to Air Ratio or Air Fuel Ratio (AFR)
FHV = 40007.2               # [kJ/kg] Fuel Heating Value (FHV)
combustor_eff = 0.99
combustor_loss = 0.018      # Combustor pressure loss
RTDF = 0.05                 # Radial Temperature Distribution Factor (RTDF)
# ------------------------------------------------------------------------------
# Turbine
hpt_eff = 0.83              # HPT Target Efficiency->Given range 0.83-0.85
hpt_vane_cooling = 0.03     # HPT Vane Cooling Air
hpt_disk_cooling = 0.165    # HPT Disk Cooling Air
lpt_eff = 0.91              # LPT Target Efficiency
hpt_vane_cooling = 0.011    # LPT Vane Cooling Air
ITD_loss = 0.006            # ITD? Loss
pt_eff = 0.92               # PT Target Efficiency
pt_disk_cooling = 0.0125    # PT disk cooling air
# ------------------------------------------------------------------------------
# Exhaust
exhaust_loss = 0.02         # Exhaust Loss
exhaust_mach = 0.15         # Exhaust Mach Number
```

Functions for calculations

In [ ]:
```
def calc_turbine_pressure(P_i, T_03, T_04, eta_turbine):
    """
    This function calculation the exit pressure of a turbine using the isentropic efficiency.
    Inputs: P_i, T_03, T_04, eta_turbine
    Outputs: P_f

    """
    temp_val_1 = 1.33333/(1.33333 - 1)
    temp_val_2 = 1 - ((T_03 - T_04) / (eta_turbine * T_03))
    P_f = P_i * (temp_val_2**temp_val_1)
    print(P_f)
    return P_f

def calc_psi(c_p, delta_T_0_s, U):
    """
    This function calculates the blade loading coefficient

    ____________
    Input: c_p, delta_T_0_s, U
    """
    psi = (2 * c_p * delta_T_0_s)/(U**2)
    print("psi = ", psi)
    return psi

def calc_lambda(alpha, phi, psi):
    """
    This function calculates the degree of reaction

    ____________
    Input: Swirl angle "alpha", flow coefficient "phi", blade loading coefficient "psi"

    ____________
    Output: Returns the value of the degree of coefficient and the tan_beta_3
    """
    alpha = math.radians(alpha)
    tan_beta_3 = math.tan(alpha) + (1/phi)
    lambda_val = (4 * tan_beta_3 * phi - psi)/4
    print("lambda = ", lambda_val)
    return lambda_val

def calc_T_static(T_total, M):
    T_static = T_total/(1 + 0.15*(M**2))
```

```
    return T_static


def calc_area(T_static, p_static, C_a, n):
    """
    This function calculate air properties and also the area from mass flow rate
    ____________
    Input: Total Temperature "T_01", Total Pressure "p_01", Flow velocity "c", axial component of the velocity "c_a", station
    ____________
    Output: A plot showing stability in real and imaginary axis.
    """
    rho_ = p_static/(R*T_static)
    A = mass_flow_rate/(rho_* C_a)/1000
    print("rho_",n, " = ", rho_ * 1000)
    print("A_",n, " = ", A)
    return A

def calc_height(A, r_m, n):
    """
    This calculated geometric values related to the cross section of the turbine
    ____________
    Input: area "A", station number "n"
    ____________
    Output: A plot showing stability in real and imaginary axis.
    """
    h = (revs * A)/(U)
    rtrm = (r_m + 0.5*h)/(r_m - 0.5*h)
    print("h_",n,"=", h)
    print("rtrm_",n,"=", rtrm)
    return h

def calc_freevortex_nozzle(r_m, r_r, r_t, alpha):
    tan_alpha_r_2 = (r_m / r_r) * math.tan(math.radians(alpha))
    tan_alpha_t_2 = (r_m / r_t) * math.tan(math.radians(alpha))

    alpha_2_r_fv = np.arctan(tan_alpha_r_2)
    alpha_2_t_fv = np.arctan(tan_alpha_t_2)
    print("nozzle alpha values: root, tip")
    print(math.degrees(alpha_2_r_fv), math.degrees(alpha_2_t_fv))

    beta_2_r_fv = math.degrees(np.arctan(tan_alpha_r_2 - ((r_m / r_r) * (1/phi))))
    beta_2_t_fv = math.degrees(np.arctan(tan_alpha_t_2 - ((r_m / r_t) * (1/phi))))
    print("nozzle beta values: root, tip")
    print(beta_2_r_fv, beta_2_t_fv)

    return math.degrees(alpha_2_r_fv), beta_2_r_fv

def calc_freevortex_rotor(r_m, r_r, r_t, alpha):
    tan_alpha_r_3 = (r_m / r_r) * math.tan(math.radians(alpha))
    tan_alpha_t_3 = (r_m / r_t) * math.tan(math.radians(alpha))

    alpha_3_r_fv = math.degrees(np.arctan(tan_alpha_r_3))
    alpha_3_t_fv = math.degrees(np.arctan(tan_alpha_t_3))
    print("rotor alpha values: root, tip")
    print(alpha_3_r_fv, alpha_3_t_fv)

    beta_3_r_fv = math.degrees(np.arctan(tan_alpha_r_3 + ((r_m / r_r) * (1/phi))))
    beta_3_t_fv = math.degrees(np.arctan(tan_alpha_t_3 + ((r_m / r_t) * (1/phi))))
    print("rotor beta values: root, tip")
    print(abs(beta_3_r_fv), abs(beta_3_t_fv))

    return alpha_3_r_fv, beta_3_r_fv
```

Cycle Calculations

Inlet

In [ ]:
```
# ================================================================================
# CYCLE CALCULATIONS
# ================================================================================
# Conditions at the Inlet
P_a = P_0 = 101.325        # [kPa]
T_a = T_0 = 296.483        # [K]

T_01 = T_0
P_01 = 0.99*P_0
print(P_01)
```

```
100.31175
```

Low Pressure Compressor

In [ ]:
```
# LPC Calculations

P_02 = lpc_pr * P_01        # Using provided pressure ratio
T_02 = T_01 + (T_01/lpc_eff * (lpc_pr**(0.285714) - 1))
```

```
W_lpc = m_dot * c_p_air * (T_02 - T_01)         # [kJ/kg]
print(P_02, T_02, W_lpc/m_dot)
```

```
401.247 463.059728696695 167.40961234017846
```

High Pressure Compressor

In [ ]:
```
# HPC Calculations

P_03 = hpc_pr * P_02
T_03 = T_02 + (T_02/hpc_eff * (hpc_pr**(0.285714) - 1))
W_hpc = m_dot * c_p_air * (T_03 - T_02)         # [kJ/kg]
print(P_03, T_03, W_hpc/m_dot)
```

```
1203.741 662.7644873827279 200.70328247946304
```

Combustion Chamber

In [ ]:
```
# Combustion Calculations

m_air = 0.91 * m_dot       # Mass flow into combustor/turbine | See next line:
# Turbine cooling air percentage can be considered as percent
# flow of turbine inlet flow.
P_04 = P_03 * 0.982
print(P_04)
T_04 = ((c_p_air * T_03) + (AFR * FHV * combustor_eff))/((1 + AFR) * c_p_gas)
print(T_04)
```

```
1182.073662
1245.3208220773054
```

High Pressure Turbine

In [ ]:
```
# HPT Calculations

m_turbine = m_air + (0.02 * m_air)      # Includes the mass of the fuel as well
m_cool_vane_hpt = 0.03 * m_turbine          # HPT Vane Cooling Air MFR
m_cool_disc_hpt = 0.0165 * m_turbine        # HPT Disk Cooling Air MFR

# Calculation of cooling after stator
T_hpt_after_vane = ((m_turbine * c_p_gas * T_04) + (m_cool_vane_hpt * c_p_air * T_03)) / (c_p_gas * (m_turbine + m_cool_vane_h
print(T_hpt_after_vane)
# Calculation after rotor but before disc cooling
T_hpt_required_energy = T_04 - ((1.01 * W_hpc) / ((m_turbine + m_cool_vane_hpt) * c_p_gas))

# Calculation after disc cooling
T_hpt_after_rotor = (((m_turbine + m_cool_vane_hpt) * c_p_gas * T_hpt_required_energy) + (m_cool_disc_hpt * c_p_air * T_03)) /
T_05 = T_hpt_after_rotor
print(T_05)
# Pressure
P_05 = calc_turbine_pressure(P_04, T_04, T_05, 0.84)
```

```
1225.9485919279928
1053.0511295978204
524.5947533586902
```

Low Pressure Turbine

In [ ]:
```
# LPT Calculations

m_turbine_lpt = m_turbine + m_cool_vane_hpt + m_cool_disc_hpt
m_cool_disc_lpt = 0.011 * m_turbine

# Calculation of work done
T_lpt_required_energy = T_05 - ((1.01 * W_lpc) / (m_turbine_lpt * c_p_gas))
print(T_lpt_required_energy)
# Calculation after disc cooling
T_lpt_after_rotor = ((m_turbine_lpt * c_p_gas * T_lpt_required_energy) + (m_cool_disc_lpt * c_p_air * T_03)) / (c_p_gas * (m_t
T_06 = T_lpt_after_rotor
print(T_06)
# Pressure
P_06 = calc_turbine_pressure(P_05, T_05, T_06, lpc_eff)
```

```
901.4232437286498
898.0819933981012
248.8071191334912
```

Exhaust

In [ ]:
```
# Exhaust Calculations
P_08 = P_0 * ((1 + ((gamma_g-1)/(2)) * exhaust_mach**2 )**(gamma_g/(gamma_g-1)))
print(P_08)
P_07 = 1.02 * P_08      # Power Turbine Exit Total Pressure
print(P_07)
```

```
102.85344186914772
104.91051070653067
```

Power Pressure Turbine

In [ ]:
```
# Power Turbine Calculations
m_turbine_pt = m_turbine_lpt + m_cool_disc_lpt
m_cool_disc_pt = 0.0125 * m_turbine
# Calculation of work done
P_06_PT = (1 - ITD_loss) * P_06
pt_pr = P_06_PT / P_07
T_pt_required_energy = T_06 - (pt_eff * T_06 * (1 - (1 / pt_pr)**((gamma_g - 1)/gamma_g)))

# PT Temperature after disc cooling
T_pt_after_rotor = ((m_turbine_pt * c_p_gas * T_pt_required_energy) + (m_cool_disc_pt * c_p_air * T_03)) / (c_p_gas * (m_turbi
T_07 = T_pt_after_rotor         # Total temperature at power turbine exit

print(P_06_PT, T_05, T_06, T_pt_required_energy,T_07)
```

```
247.31427641869024 1053.0511295978204 898.0819933981012 738.6486734668399 736.7977261944576
```

Work and SFC Calculations

In [ ]:
```
# Calculation of work
W_pt = c_p_gas * (T_06 - T_pt_required_energy) * 0.99
SFC = (3600 * AFR) / W_pt
print(W_pt, SFC)
```

```
181.19915676827705 0.397352842497362
```

Appendix B

HPT Calculation Functions

Space left blank on purpose.

Gas Turbine Design Analysis Functions

In [ ]:
```
import numpy
import numpy as np
import math
import pandas as pd
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

mechanical_eff = 0.99
gamma_air = 1.4
gamma_g = 1.33333
c_p_air = 1.005
c_p_gas = 1.148
R = 0.287

m_cool_vane_hpt = 0.1450776
m_cool_disc_hpt = 0.0797

# TURBINE INLET
T_01 = 1245.3208220773054        # [k]
P_01 = 1182.073662         # [kPa]
m_dot_1 = 4.835922
M_1 = 0.125

T_02_cycle = 1225.9485919279928

# BETWEEN STATOR AND ROTOR
m_dot_2 = m_dot_1 + m_cool_vane_hpt

# TURBINE EXIT
T_03 = 1041.2535775588708
T_03_cooled = 1033.9843383376274
P_03 = 517.9223058003336
m_dot_3 = m_dot_2 + m_cool_disc_hpt


# BLADE PARAMETERS
# VANE
AR_vane = 0.5
TE_vane = 1.27/1000     # minimum trailing edge thickness in [m]
LE_diameter_vane = 0.000508

# BLADE
AR_rotor = 1.3
TE_rotor = 0.762/1000   # minimum trailing edge thickness in [m]
tip_clearance = 2.0    # minimum tip clearance span -> maximum is 0.02
LE_diameter_rotor = 0.00508             #0.000254*12

"""
This file contains all the main input and functions
"""

class aeroturbine():

    def calc_properties(M, T_stagnation, P_stagnation):
        T = T_stagnation/(1 + ((gamma_g - 1)/2) * M**2)
        P = P_stagnation/((1 + ((gamma_g - 1)/2) * M**2))**(gamma_g/(gamma_g - 1))
        rho = P/(0.287*T)
        c = M * numpy.sqrt(gamma_g * 287 * T) #--

        return T, P, rho, c

    def calc_U(psi):
        """
        This function calculates the metal speed U.
        Input:  Stage loading coefficient "psi"
        Output: Returns the value of U
        """
        U = numpy.sqrt((2*c_p_gas*1000*(T_02_cycle - T_03)) / (psi))
        return U

    def calc_stage_3(U, C_3, T_3, rho_3,P_3,alpha_3):

        #V_w_3 = c_p_gas*1000*(T_01-T_03)/(2*U) + reaction*U
        C_a_3 = C_3 * np.cos(np.radians(alpha_3))
        C_w_3 = math.sqrt(C_3**2 - C_a_3**2)
        V_w_3 = U + C_w_3
        alpha_3 = numpy.rad2deg(numpy.arcsin(C_w_3/C_3))
        V_3 = np.sqrt(V_w_3**2 + C_a_3**2)
        flow_coefficient_3 =  C_a_3 / U
        beta_3 = np.rad2deg(np.arctan(V_w_3 / C_a_3))
        a_3 = np.sqrt(gamma_g * R * 1000 * T_3)
        M_3_rel = V_3 / a_3
        A_3 = m_dot_3/(rho_3 * C_a_3)
```

```
    P_03_rel = P_3*(1+ (gamma_g-1)/2 * M_3_rel**2)**(gamma_g/(gamma_g-1))

    return C_a_3, C_w_3,  V_3, V_w_3, flow_coefficient_3, beta_3, a_3, M_3_rel, A_3, P_03_rel

def calc_stage_2(U, reaction, T_1, T_3, P_3, A_3, V_w_3):
    T_2 = T_3 + reaction * (T_1 - T_3)
    P_2 = P_3 * ((T_2/T_3) ** (gamma_g/(gamma_g - 1)))
    rho_2 = P_2 / (R * T_2)
    A_2 = A_3
    C_a_2 = (m_dot_2)/(rho_2 * A_2)
    flow_coefficient_2 = C_a_2 / U
    a_2 = np.sqrt(gamma_g * R * 1000 * T_2)
    # V_w_2 = V_w_3 - (2*reaction * U)
    V_w_2 = (c_p_gas * 1000 * (T_01 - T_03) / (U)) - V_w_3 #This is the connection to work -> we should change this to T_02 maybe
    beta_2 = np.rad2deg(np.arctan(V_w_2 / C_a_2))
    V_2 = np.sqrt(V_w_2**2 + C_a_2**2)
    C_w_2 = (V_w_2 + U)
    C_2 = np.sqrt(C_w_2**2 + C_a_2**2)
    alpha_2 = np.rad2deg(np.arctan(C_w_2/C_a_2))
    M_2 = C_2 / a_2
    M_2_rel = V_2 / a_2
    P_02 = P_2*(1+ (gamma_g-1)/2 * M_2**2)**(gamma_g/(gamma_g-1))
    P_02_rel = P_2*(1+ (gamma_g-1)/2 * M_2_rel**2)**(gamma_g/(gamma_g-1))
    T_02 = T_2 + C_2**2/(2*1000*c_p_gas)



    return T_02, T_2, P_2, rho_2, A_2, C_a_2, flow_coefficient_2, a_2, V_w_2, beta_2, V_2, C_w_2, C_2, alpha_2, M_2, M_2_rel,P_02,P_02_

def calc_stage_2_trial(U,T_1,T_3,P_3,A_3,V_w_3):

    error = 1
    max_iterations = 10000
    count = 0
    T_2 = 1060
    increment = 0.01
    V_w_2 = (c_p_gas * 1000 * (T_02_cycle - T_03) / (U)) - V_w_3
    C_w_2 = (V_w_2 + U)

    while count < max_iterations:

        P_2 = P_01 * ((T_2/T_02_cycle) ** (gamma_g/(gamma_g - 1))) #changed P_2 = P_3 * ((T_2/T_3)**(gamma_g/(gamma_g-1)))
        rho_2 = P_2/(R*T_2)
        T_02 = T_2 + (C_w_2**2 + (m_dot_2/(rho_2*A_3))**2)/(2*c_p_gas*1000)
        #print(T_02)
        error = np.abs(T_02_cycle - T_02) #cycle_Calc T_02 to be defined as global constant
        count = count + 1
        if (0 < error < 0.001):
            break
        else:
            T_2 = T_2 + increment

    if count != max_iterations:
        reaction = (T_2 - T_3)/(T_1 - T_3)
        A_2 = A_3
        C_a_2 = (m_dot_2)/(rho_2 * A_2)

        flow_coefficient_2 = C_a_2 / U
        a_2 = np.sqrt(gamma_g * R * 1000 * T_2)

        beta_2 = np.rad2deg(np.arctan(V_w_2 / C_a_2))
        V_2 = np.sqrt(V_w_2**2 + C_a_2**2)

        C_2 = np.sqrt(C_w_2**2 + C_a_2**2)
        alpha_2 = np.rad2deg(np.arctan(C_w_2/C_a_2))
        M_2 = C_2 / a_2
        M_2_rel = V_2 / a_2
        P_02 = P_2*(1+ (gamma_g-1)/2 * M_2**2)**(gamma_g/(gamma_g-1))
        P_02_rel = P_2*(1+ (gamma_g-1)/2 * M_2_rel**2)**(gamma_g/(gamma_g-1))

    else:
        T_02 = 0
        T_2 = 0
        P_2 = 0
        rho_2 = 0
        A_2 = 0
        C_a_2 = 0
        flow_coefficient_2 = 0
        a_2 = 0
        V_w_2 = 0
        beta_2 = 0
        V_2 = 0
        C_w_2 = 0
        C_2 = 0
        alpha_2 = 0
        M_2 = 0
        M_2_rel = 0
        P_02 = 0
        P_02_rel = 0
        reaction = 0
```

```
        return T_02, T_2, P_2, rho_2, A_2, C_a_2, flow_coefficient_2, a_2, V_w_2, beta_2, V_2, C_w_2, C_2, alpha_2, M_2, M_2_rel,P_02,P_02_

    def calc_hub_angles(r_m_pointer, r_hub_pointer, alpha_2_pointer, alpha_3_pointer, flow_coeff_2_pointer, flow_coeff_3_pointer, U_pointer
        alpha_2_hub_rad = numpy.arctan((r_m_pointer/r_hub_pointer) *numpy.tan(numpy.deg2rad(alpha_2_pointer)))
        alpha_2_hub_deg = numpy.rad2deg(alpha_2_hub_rad)

        alpha_3_hub_rad = numpy.arctan((r_m_pointer/r_hub_pointer) *numpy.tan(numpy.deg2rad(alpha_3_pointer)))
        alpha_3_hub_deg = numpy.rad2deg(alpha_3_hub_rad)

        beta_2_hub_rad = numpy.arctan((r_m_pointer/r_hub_pointer) *numpy.tan(numpy.deg2rad(alpha_2_pointer)) - (r_hub_pointer/r_m_pointer)*
        beta_2_hub_deg = numpy.rad2deg(beta_2_hub_rad)

        beta_3_hub_rad = numpy.arctan((r_m_pointer/r_hub_pointer) *numpy.tan(numpy.deg2rad(alpha_3_pointer)) + (r_hub_pointer/r_m_pointer)*
        beta_3_hub_deg = numpy.rad2deg(beta_3_hub_rad)

        U_hub = U_pointer * (r_hub_pointer / r_m_pointer)

        V_2_hub = C_a_2/np.cos(beta_2_hub_rad)
        C_2_hub = C_a_2/np.cos(alpha_2_hub_rad)
        C_3_hub = C_a_3/np.cos(alpha_3_hub_rad)

        T_2_hub = T_02 - (C_2_hub**2)/(2*c_p_gas*1000)
        T_3_hub = T_03 - (C_3_hub**2)/(2*c_p_gas*1000)
        T_1_hub = T_1 #assumed since no free vortexing

        a_2_hub = np.sqrt(gamma_g*T_2_hub*R*1000)

        M_2_rel_hub = V_2_hub / a_2_hub
        M_2_hub = C_2_hub / a_2_hub

        reaction_hub = (T_2_hub-T_3_hub)/(T_1_hub-T_3_hub)

        return alpha_2_hub_deg, alpha_3_hub_deg, beta_2_hub_deg, beta_3_hub_deg,  U_hub, V_2_hub, C_2_hub, M_2_rel_hub, M_2_hub,reaction_hul



    def calc_tip_angles(r_m_pointer, r_tip_pointer, alpha_2_pointer, alpha_3_pointer, flow_coeff_2_pointer, flow_coeff_3_pointer, U_pointer
        alpha_2_tip_rad = numpy.arctan((r_m_pointer/r_tip_pointer) *numpy.tan(numpy.deg2rad(alpha_2_pointer)))
        alpha_2_tip_deg = numpy.rad2deg(alpha_2_tip_rad)

        alpha_3_tip_rad = numpy.arctan((r_m_pointer/r_tip_pointer) *numpy.tan(numpy.deg2rad(alpha_3_pointer)))
        alpha_3_tip_deg = numpy.rad2deg(alpha_3_tip_rad)

        beta_2_tip_rad = numpy.arctan((r_m_pointer/r_tip_pointer) *numpy.tan(numpy.deg2rad(alpha_2_pointer)) - (r_tip_pointer/r_m_pointer)*
        beta_2_tip_deg = numpy.rad2deg(beta_2_tip_rad)

        beta_3_tip_rad = numpy.arctan((r_m_pointer/r_tip_pointer) *numpy.tan(numpy.deg2rad(alpha_3_pointer)) + (r_tip_pointer/r_m_pointer)*
        beta_3_tip_deg = numpy.rad2deg(beta_3_tip_rad)

        U_tip = U_pointer * (r_tip_pointer / r_m_pointer)
        V_2_tip = C_a_2/np.cos(beta_2_tip_rad)
        C_2_tip = C_a_2/np.cos(alpha_2_tip_rad)

        C_2_tip = C_a_2/np.cos(alpha_2_tip_rad)
        C_3_tip = C_a_3/np.cos(alpha_3_tip_rad)
        V_3_tip = C_a_3/np.cos(beta_3_tip_rad)

        T_2_tip = T_02 - (C_2_tip**2)/(2*c_p_gas*1000)
        T_3_tip = T_03 - (C_3_tip**2)/(2*c_p_gas*1000)


        a_2_tip = np.sqrt(gamma_g*T_2_tip*R*1000)
        a_3_tip = np.sqrt(gamma_g*T_3_tip*R*1000)

        M_2_rel_tip = V_2_tip / a_2_tip
        M_2_tip = C_2_tip / a_2_tip
        M_3_rel_tip = V_3_tip / a_3_tip

        M_2_rel_tip = V_2_tip / a_2
        M_2_tip = C_2_tip / a_2


        return alpha_2_tip_deg, alpha_3_tip_deg, beta_2_tip_deg, beta_3_tip_deg, U_tip, V_2_tip, C_2_tip, M_2_rel_tip, M_2_tip, M_3_rel_tip


    def calc_tip_hub_reaction(c_a_3_pointer, c_a_2_pointer ,beta_2_hub, beta_2_tip, beta_3_hub, beta_3_tip, U_hub, U_tip):

        reaction_hub = (c_a_3_pointer * numpy.tan(numpy.deg2rad(beta_3_hub)) - c_a_2_pointer*numpy.tan(numpy.deg2rad(beta_2_hub)))/(2*U_hub
        reaction_tip = (c_a_3_pointer * numpy.tan(numpy.deg2rad(beta_3_tip)) - c_a_2_pointer*numpy.tan(numpy.deg2rad(beta_2_tip)))/(2*U_tip
        return reaction_hub, reaction_tip


class aerostructural():
    def calc_structural(an_squared_pointer, area_2_pointer, U_meanline_pointer):
        N = numpy.sqrt((an_squared_pointer)/area_2_pointer)
        omega = N*2*numpy.pi/60
        r_meanline = U_meanline_pointer/omega
        h = (area_2_pointer * (N/60))/U_meanline_pointer
        r_hub = r_meanline - (h/2)
        r_tip = r_meanline + (h/2)
        return N, omega, r_hub, r_tip, r_meanline, h
```

```python
class aerodynamic_losses():
    """
    Profile losses calculated in this class.
    """
    class profile_losses():
        """
        Profile losses calculated in this class.
        """
        def figure_2_3a(pitch_chord_ratio, exit_flow_angle):
            fig_2_3a = pd.read_csv(r'_input_database\figure_2_3a.csv')
            X = fig_2_3a[['pitch_chord_ratio', 'exit_flow_angle']]
            y = fig_2_3a['K_P_1']

            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

            model = LinearRegression()
            model.fit(X_train, y_train)

            X = pd.DataFrame({'pitch_chord_ratio': [pitch_chord_ratio], 'exit_flow_angle': [exit_flow_angle]})
            K_P_1 = model.predict(X)

            return K_P_1[0]

        def figure_2_3b(pitch_chord_ratio, exit_flow_angle):
            fig_2_3b = pd.read_csv(r'_input_database\figure_2_3b.csv')
            X = fig_2_3b[['pitch_chord_ratio', 'exit_flow_angle']]
            y = fig_2_3b['K_P_2']

            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

            model = LinearRegression()
            model.fit(X_train, y_train)

            X = pd.DataFrame({'pitch_chord_ratio': [pitch_chord_ratio], 'exit_flow_angle': [exit_flow_angle]})
            K_P_2 = model.predict(X)

            return K_P_2[0]

        def figure_2_4(beta_b1, beta_b2):
            beta_eff = beta_b1 + beta_b2
            fig_2_4 = pd.read_csv(r'_input_database\figure_2_4.csv')
            X = fig_2_4[['beta_b1_b2']]
            y = fig_2_4['tmax_and_c']

            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

            model = LinearRegression()
            model.fit(X_train, y_train)

            X = pd.DataFrame({'beta_b1_b2': [beta_eff]})
            tmax_and_c = model.predict(X)

            return tmax_and_c[0]

        def figure_2_5(beta_b1, beta_b2):
            fig_2_5 = pd.read_csv(r'_input_database\figure_2_5.csv')
            X = fig_2_5[['beta_b1', 'beta_b2']]
            y = fig_2_5['Stagger Angle']

            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

            model = LinearRegression()
            model.fit(X_train, y_train)

            X = pd.DataFrame({'beta_b1': [beta_b1], 'beta_b2': [beta_b2]})
            stagger_angle = model.predict(X)

            return stagger_angle[0]


        def figure_2_6(x_value, x=True):
            """
            x = True for Rotor
            """
            fig_2_6 = pd.read_csv(r'_input_database\figure_2_6.csv')
            rhrt = fig_2_6['x'].values
            y1 = fig_2_6['Rotor'].values
            y2 = fig_2_6['Nozzle'].values

            if x == True:
                interp_func = interp1d(rhrt, y1, kind='cubic')
                interpolated_y = interp_func(x_value)
            else:
                interp_func = interp1d(rhrt, y2, kind='cubic')
                interpolated_y = interp_func(x_value)

            return interpolated_y

        def figure_2_7(x_value):
```

```
        fig_2_7 = pd.read_csv(r'_input_database\figure_2_7.csv')
        x = fig_2_7['M_1_hub'].values
        y = fig_2_7['delta_p_q_1_hub'].values

        interp_func = interp1d(x, y, kind='cubic')
        interpolated_y = interp_func(x_value)

        return interpolated_y


    """
    The following section give information about the symbols to be used for the following class.
    This is established from the information provided in Axial and Radial Turbines Part A - Moustapha.
    """
    def calc_K_p(M_1, M_2, M_1_rel_hub, P_1, P_2 ,r_tip, r_hub, beta_in, beta_out, zweifel_pointer):
        """
        Using corrected effects of exit mach number.
        Function used to determine K_accel.
        Equation (2.7), (2.8)
        IMPORTANT:
        FOR STATOR
        beta_in -> alpha_1
        beta_out -> alpha_2
        FOR ROTOR
        beta_in -> beta_2
        beta_out -> beta_3

        """
        #  ============= K_p_star =============
        # get the stagger angle from figure 2.5
        stagger_angle = aerodynamic_losses.profile_losses.figure_2_5(beta_in, beta_out)      # [deg]
        tmax_and_c = aerodynamic_losses.profile_losses.figure_2_4(beta_in, beta_out)

        pitch_chord_ratio = (zweifel_pointer / (2 * (numpy.tan(numpy.radians(beta_in)) + numpy.tan(numpy.radians(beta_out)) ) * (numpy.
        pitch_axial_chord_ratio = pitch_chord_ratio/numpy.cos(numpy.radians(stagger_angle))
        K_P_2 = aerodynamic_losses.profile_losses.figure_2_3b(pitch_chord_ratio, beta_out)         # beta_3 in other code
        K_P_1 = aerodynamic_losses.profile_losses.figure_2_3a(pitch_chord_ratio, beta_out)         # beta_3 in other code
        K_p_star = (K_P_1 + (abs(beta_in / beta_out) * (beta_in / beta_out))*(K_P_2 - K_P_1)) * ((tmax_and_c / 0.2)**(beta_in / beta_out
        #  =============== K_sh ===============
        k = 1.333333

        if M_1_rel_hub <= 0.4:
            A = 0
        if M_1_rel_hub > 0.4:
            A = 0.75 * ((M_1_rel_hub - 0.4)**(7/4))

        B = A * (r_hub/r_tip)
        C = 1 - ((1 + ((k - 1) / (2)) * M_1**2)**(k/(k - 1)))
        D = 1 - ((1 + ((k - 1) / (2)) * M_2**2)**(k/(k - 1)))

        K_sh = B * (P_1 / P_2) * (C / D)

        #  ============== K_accel ==============
        if M_2 <= 0.2:
            K_1 = 1.0
        if M_2 > 0.2:
            K_1 = 1 - 1.25 * (M_2 - 0.2)
        K_2 = (M_1/M_2)**2
        K_accel = 1 - K_2 * (1 - K_1)

        #  ================ K_p ================
        K_p = 0.914 * ((2/3) * K_p_star * K_accel + K_sh)

        return K_p, pitch_chord_ratio, K_accel, stagger_angle,  pitch_chord_ratio, pitch_axial_chord_ratio, K_1, K_2

class secondary_losses():

    def calc_K_s(K_accel, AR, beta_in, beta_out):
        """
        FOR STATOR
        beta_in -> alpha_1
        beta_out -> alpha_2
        FOR ROTOR
        beta_in -> beta_2
        beta_out -> beta_3
        """

        if AR <= 2:
            f_AS = (1- 0.25 * np.sqrt(2 - AR))/(AR)
        if AR > 2:
            f_AS = 1/AR
        alpha_m = np.arctan(0.5 * (np.tan(np.radians(beta_in)) - np.tan(np.radians(beta_out))))
        A = np.cos(np.radians(beta_out))/np.cos(np.radians(beta_in))
        B = 2 * (np.tan(np.radians(beta_in)) + np.tan(np.radians(beta_out))) * np.cos(alpha_m)
        C = ((np.cos(np.radians(beta_out)))**2) / ((np.cos(alpha_m))**3)
        K_s_star = 0.0334 * f_AS * A *  (B**2) * C
        K_3 = 1 / ((AR)**2)
        K_cs = 1 - K_3 * (1 - K_accel)
        K_s = 1.2 * K_s_star * K_cs
```

```
        return K_s

class trailing_edge_losses_rotor():

    def figure_2_10(x_value, beta_in, beta_out):
        """
        f -> delta_phi_squared_TE
        FOR STATOR
        beta_in -> alpha_1
        beta_out -> alpha_2
        FOR ROTOR
        beta_in -> beta_2
        beta_out -> beta_3
        """
        fig_2_10 = pd.read_csv(r'_input_database\figure_2_10.csv')
        r_te_o = fig_2_10['r_te_o'].values
        impulse_blading = fig_2_10['impulse_blading'].values
        stator_vanes = fig_2_10['stator_vanes'].values

        interp_func_1 = interp1d(r_te_o, impulse_blading, kind='cubic')
        interp_func_2 = interp1d(r_te_o, stator_vanes, kind='cubic')
        f_alpha  = interp_func_1(x_value)
        f_zero = interp_func_2(x_value)

        f = f_zero + (abs(beta_in / beta_out) * (beta_in / beta_out))*(f_alpha - f_zero)

        return f


    def required_vals(h, stagger_angle, r_meanline, pitch_axial_chord_ratio, beta_3):
        c_true = (h)/AR_rotor
        c_a = (h * np.cos(np.radians(stagger_angle)))/AR_rotor
        N = math.floor((2 * np.pi * r_meanline) /(pitch_axial_chord_ratio * c_a))
        o = (pitch_axial_chord_ratio * c_a) * np.cos(np.radians(beta_3))

        return c_true, c_a, N, o


    def K_TET(M_2, beta_in, beta_out, h, stagger_angle, r_meanline, pitch_axial_chord_ratio):
        """
        M_2 -> Use relative M_2_rel for the rotor.
        FOR STATOR
        beta_in -> alpha_1
        beta_out -> alpha_2
        FOR ROTOR
        beta_in -> beta_2
        beta_out -> beta_3
        """
        c_true, c_a, N, o = aerodynamic_losses.trailing_edge_losses_rotor.required_vals(h, stagger_angle, r_meanline, pitch_axial_chord_


        r_to_o = TE_rotor/o
        f = aerodynamic_losses.trailing_edge_losses_rotor.figure_2_10(r_to_o ,beta_in, beta_out)

        term1 = ((gamma_g - 1) / 2) * M_2**2
        term2 = (1 / (1 - f)) - 1
        numerator = ((1 - term1 * (term2))**(-gamma_g / (gamma_g - 1)) )- 1
        denominator = 1 - ((1 + term1))**(-gamma_g / (gamma_g - 1))
        K_TE = numerator / denominator
        throat_opening = o

        return K_TE, N, c_true, c_a, throat_opening

class trailing_edge_losses_stator():

    def figure_2_10(x_value, beta_in, beta_out):
        """
        f -> delta_phi_squared_TE
        FOR STATOR
        beta_in -> alpha_1
        beta_out -> alpha_2
        FOR ROTOR
        beta_in -> beta_2
        beta_out -> beta_3
        """
        fig_2_10 = pd.read_csv(r'_input_database\figure_2_10.csv')
        r_te_o = fig_2_10['r_te_o'].values
        impulse_blading = fig_2_10['impulse_blading'].values
        stator_vanes = fig_2_10['stator_vanes'].values

        interp_func_1 = interp1d(r_te_o, impulse_blading, kind='cubic')
        interp_func_2 = interp1d(r_te_o, stator_vanes, kind='cubic')
        f_alpha  = interp_func_1(x_value)
        f_zero = interp_func_2(x_value)

        f = f_zero + (abs(beta_in / beta_out) * (beta_in / beta_out))*(f_alpha - f_zero)

        return f


    def required_vals(h, stagger_angle, r_meanline, pitch_axial_chord_ratio, beta_3):
```

```
        c_true = (h)/AR_vane
        c_a = (h * np.cos(np.radians(stagger_angle)))/AR_vane
        N = math.floor((2 * np.pi * r_meanline) /(pitch_axial_chord_ratio * c_a))
        o = (pitch_axial_chord_ratio * c_a) * np.cos(np.radians(beta_3))

        return c_true, c_a, N, o


    def K_TET(M_2, beta_in, beta_out, h, stagger_angle, r_meanline, pitch_axial_chord_ratio):
        """
        M_2 -> Use relative M_2_rel for the rotor.
        FOR STATOR
        beta_in -> alpha_1
        beta_out -> alpha_2
        FOR ROTOR
        beta_in -> beta_2
        beta_out -> beta_3
        """
        c_true, c_a, N, o = aerodynamic_losses.trailing_edge_losses_stator.required_vals(h, stagger_angle, r_meanline, pitch_axial_chor


        r_to_o = TE_vane/o
        f = aerodynamic_losses.trailing_edge_losses_stator.figure_2_10(r_to_o ,beta_in, beta_out)

        term1 = ((gamma_g - 1) / 2) * M_2**2
        term2 = (1 / (1 - f)) - 1
        numerator = ((1 - term1 * (term2))**(-gamma_g / (gamma_g - 1)) )- 1
        denominator = 1 - ((1 + term1))**(-gamma_g / (gamma_g - 1))
        K_TE = numerator / denominator
        throat_opening = o
        return K_TE, N, c_true, c_a, throat_opening


def efficiency_calculations(K_stator, K_rotor, M_2, M_3_rel, C_2, V_3):
    zeta_N = K_stator/(1 + 0.5 * gamma_g * M_2**2)
    zeta_R = K_rotor/(1 + 0.5 * gamma_g * M_3_rel**2)
    eta_tt = 1 / (1 + ((zeta_N * C_2**2 + zeta_R * V_3**2) / (2 * c_p_gas * 1000 * (T_01 - T_03))))
    eta_tt = eta_tt * 100
    return eta_tt

def efficiency_final(eta_tt, h, beta_3, r_tip, r_meanline):
    delta_n = 0.93 * (eta_tt/100) * ((tip_clearance/100)/(h * np.cos(np.radians(beta_3)))) * (r_tip/r_meanline)
    eta_final = eta_tt - delta_n
    return delta_n, eta_final

def losses_off_design(K_p_rotor, K_s_rotor, K_stator, K_1_rotor, K_2_rotor, pitch_chord_ratio_rotor, pitch_axial_chord_ratio_rotor, c_t
    # Primary
    phi_squared_P0 = 1 / (1 + ((K_p_rotor) / (K_1_rotor + K_2_rotor * K_p_rotor)))
    s = pitch_chord_ratio_rotor * c_true
    d_s = LE_diameter_rotor/s
    graph_x = (d_s)**(-1.6) * (np.cos(np.radians(beta_2)) / np.cos(np.radians(beta_3)))**(-2) * (np.radians(incidence))

    def figure_2_34(graph_x):
        fig_2_34 = pd.read_csv(r'_input_database\figure_2_34.csv')
        x = fig_2_34['graph_x'].values
        y = fig_2_34['deg_of_accel'].values

        interp_func = interp1d(x, y, kind='cubic')
        interpolated_y = interp_func(graph_x)

        return interpolated_y

    deg_of_accel = figure_2_34(graph_x)
    phi_squared_P = phi_squared_P0 - deg_of_accel
    K_p_od = (K_1_rotor * (1-phi_squared_P)) / (phi_squared_P - K_2_rotor * (1-phi_squared_P))


    # Secondary
    d_c = LE_diameter_rotor/c_true
    graph_x_2 = (d_c)**(-0.3) * (np.cos(np.radians(beta_2)) / np.cos(np.radians(beta_3)))**(-1.5) * ((np.radians(incidence))/(np.radian
    if graph_x_2 < 0.27:
        def figure_2_35(graph_x_2):
            fig_2_35 = pd.read_csv(r'_input_database\figure_2_35.csv')
            x = fig_2_35['x'].values
            y = fig_2_35['K_K_des'].values

            interp_func = interp1d(x, y, kind='cubic')
            interpolated_y = interp_func(graph_x_2)

            return interpolated_y

        K_K_des = figure_2_35(graph_x_2)
        K_s_od =  K_K_des * K_s_rotor

        # Trailing Edge TO BE CONFIRMED
        K_TET_od, N_rotor_od, c_true_rotor_od, c_a_rotor_od,dummy = aerodynamic_losses.trailing_edge_losses_rotor.K_TET(M_2_rel_od, beta

        K_rotor_od = K_p_od + K_s_od + K_TET_od

        eta_tt_od = aerodynamic_losses.efficiency_calculations(K_stator, K_rotor_od, M_2_rel_od, M_3_rel_od, C_2, V_3)
        delta_n_od, eta_final_od = aerodynamic_losses.efficiency_final(eta_tt_od, h, beta_3, r_tip, r_meanline)
```

```
        else:
            K_K_des, K_s_od, eta_tt_od, delta_n_od, eta_final_od = 0,0,0,0,0


        return eta_tt_od, delta_n_od, eta_final_od


class off_design():
    def calc_off_design(A_3, U_mean,beta_3,Ca_2, Cw_2, beta_2, a_2, a_3):
        U_mean_od = U_mean *0.9
        C_w_2_od = Cw_2
        flow_coeff_2_od = Ca_2/U_mean_od
        V_w_2_od = Cw_2 - U_mean_od
        alpha_2_rel_od = np.arctan(V_w_2_od/Ca_2)
        alpha_2_rel_od_deg = np.rad2deg(alpha_2_rel_od)
        incidence_2 = alpha_2_rel_od_deg - beta_2 #beta_2 is the blade angle -> alpha_2_rel from previous calculations
        v_2_od = np.sqrt(V_w_2_od**2 + Ca_2**2) #relative velocity on the hypoteneuse (total relative velocity at 2)
        M_2_rel_od = v_2_od / a_2 #assumed speed of sound at 2 OD = speed of sound on design.
        T_2 = a_2**2/(gamma_g*R*1000)

        LHS = R*m_dot_3/A_3
        error_threshold = LHS * 0.01 #1 percent error of the LHS

        Ca_3_range = np.linspace(100,400,1000)
        Ca_3_od = 0

        for i in Ca_3_range:
            V_w_3_od = i * np.tan(np.deg2rad(beta_3))
            C_w_3_od = V_w_3_od - U_mean_od
            C_3_od = np.sqrt(i**2 + C_w_3_od**2)
            T_3_od = T_03 - (C_3_od**2)/(2*1000*c_p_gas)
            P_3_od = P_03 * (T_3_od/T_03)**(gamma_g/(gamma_g-1))
            RHS = i *P_3_od/T_3_od
            a_3_od = math.sqrt(gamma_g*R*1000*T_3_od)

            if np.abs(LHS - RHS) < error_threshold:
                Ca_3_od = i
                alpha_3_od = np.rad2deg(np.arctan(C_w_3_od/Ca_3_od))
                rho_3_od = P_3_od/(R*T_3_od)
                work_od_cw = U_mean_od*(C_w_3_od + C_w_2_od)
                work_od_vw = U_mean_od*(V_w_3_od + V_w_2_od)
                flow_coeff_3_od = Ca_3_od/U_mean_od
                V_3_od = np.sqrt(V_w_3_od**2 + Ca_3_od**2)
                M_3_rel_od = V_3_od/a_3_od
                # For physics check
                P_2 = P_3_od * ((T_2/T_3_od) ** (gamma_g/(gamma_g - 1)))
                P_02_rel = P_2*(1+ (gamma_g-1)/2 * M_2_rel_od**2)**(gamma_g/(gamma_g-1))
                P_03_rel = P_3_od*(1+ (gamma_g-1)/2 * M_3_rel_od**2)**(gamma_g/(gamma_g-1))
                break

        if Ca_3_od == 0: #if nothing happens, return everything as zero
            V_w_3_od =0
            C_w_3_od = 0
            C_3_od = 0
            T_3_od = 0
            P_3_od = 0
            Ca_3_od = 0
            alpha_3_od = 0
            rho_3_od = 0
            work_od_cw = 0
            work_od_vw = 0
            flow_coeff_3_od = 0
            M_3_rel_od= 0
            P_02_rel = 0
            P_03_rel = 0

            return T_3_od, rho_3_od, P_3_od, alpha_3_od, alpha_2_rel_od_deg, flow_coeff_2_od, incidence_2, v_2_od, C_w_3_od,Ca_3_od,U_mean_

        else:
            return T_3_od, rho_3_od, P_3_od, alpha_3_od, alpha_2_rel_od_deg, flow_coeff_2_od, incidence_2, v_2_od, C_w_3_od,Ca_3_od,U_mean_




def verify_zweifel_rotor(r_hub, h, N,c, alpha_1, alpha_2):
    s = (2*np.pi * (r_hub + 0.5*h))/N
    zxr = 2 * ((s/c)) * (np.tan(np.radians(alpha_1)) + np.tan(np.radians(alpha_2))) * np.cos(np.radians(alpha_2))**2
    return zxr

def verify_zweifel_stator(r_hub, h, N,c, alpha_1, alpha_2):
    s = (2*np.pi * (r_hub + 0.5*h))/N
    zxs = 2 * ((s/c)) * (np.tan(np.radians(alpha_1)) + np.tan(np.radians(alpha_2))) * np.cos(np.radians(alpha_2))**2
    return zxs
```

Appendix C

HPT Calculation Loop

Space left blank on purpose.

In [ ]:
```
import numpy
import numpy as np
import math
import pandas as pd
from scipy.optimize import minimize
import itertools
from functools import reduce
from gt_design import *
```

Define Ranges

Define the range for the stage loading coefficient, flow coefficient and the reaction. Using the obtained values, calculate the other values.

Use the following data:

1. Reaction (Λ) -> 0.35 - 0.65
2. Flow Coefficient (ϕ) -> 0.6 - 1.0
3. Structural Limit (AN^2) -> 10000000 - 22580600 (checking for a wide range)
4. Zweifel for vane -> 0.75 - 0.90
5. Zweifel for rotor -> 0.80 - 0.95

In [ ]:
```
range_stage_loading = numpy.linspace(2.5, 4.5, 15)
range_alpha_3 = numpy.linspace(10, 25, 10)
range_mach_exit = numpy.linspace(0.3, 0.55, 10)
range_AN_squared = numpy.linspace(15000000, 22580600, 20)
range_zweifel_vane = numpy.linspace(0.75, 0.90, 4)
range_zweifel_rotor = numpy.linspace(0.80, 0.95, 4)
range_incidence = numpy.linspace(2, 4, 6)
```

int main(){}

In [ ]:
```
def int_main(range_mach_exit, range_stage_loading, range_alpha_3, range_AN_squared, range_incidence, range_zweifel_vane, range_zweifel_roto
    """
    Design Point Angles:
    beta_2m -> rotor inlet metal angle
    beta_3m -> rotor exit metal angle

    Off Design Angles:
    beta_2 -> rotor inlet flow angle
    beta_3 -> rotor exit flow angle

    IMP -> FOR DESIGN POINT -> beta_2m = beta_2, beta_3m -> beta_3

    INCIDENCE -> incidence = beta_2 - beta_2m, DEVIATION (for the project) -> beta_3 = beta_3m
    """
    data_meanline = []
    data_root_hub = []
    data_meanline_losses = []
    data_off_design = []
    data_efficiency = []
    data_scratch = []

    #for i, j, k, l, m, n, o, p in itertools.product(range_mach_exit, range_stage_loading, range_alpha_3, range_degree_reaction, range_AN_sq
    for i, j, k, m, n, o, p in itertools.product(range_mach_exit, range_stage_loading, range_alpha_3, range_AN_squared, range_incidence, ra
        incidence = n

        # MEANLINE ANALYSIS
        T_1, P_1, rho_1, C_1 = aeroturbine.calc_properties(M_1, T_01, P_01)
        C_a_1 = C_1 * np.cos(np.radians(-10))
        T_3, P_3, rho_3, C_3 = aeroturbine.calc_properties(i, T_03_cooled, P_03)
        C_w_1 = numpy.sqrt(C_1**2 - C_a_1**2)

        # Calculate U from the stage loading.
        U = aeroturbine.calc_U(j)
        C_a_3, C_w_3,  V_3, V_w_3, flow_coefficient_3, beta_3m, a_3, M_3_rel, A_3,P_03_rel = aeroturbine.calc_stage_3(U, C_3, T_3, rho_3,P_
        T_02,T_2, P_2, rho_2, A_2, C_a_2, flow_coefficient_2, a_2, V_w_2, beta_2m, V_2, C_w_2, C_2, alpha_2, M_2, M_2_rel,P_02, P_02_rel,l
        if T_02 > 0:

            # STRUCTURAL ANALYSIS
            A_1 = m_dot_1/(rho_1 * C_a_1)
            N, omega, r_hub, r_tip, r_meanline, h = aerostructural.calc_structural(m, A_2, U)

            if V_3 > V_2 and C_2 > C_1 and rho_3 < rho_2 < rho_1 and T_02 < T_01 and T_02 > T_03 and T_1 > T_2 and T_2 > T_3 and P_1 > P_2

                # BLADE VORTEX ANALYSIS
                alpha_2_hub, alpha_3_hub, beta_2_hub, beta_3_hub, U_hub, V_2_hub, C_2_hub, M_2_rel_hub, M_2_hub, reaction_hub = aeroturbine
                alpha_2_tip, alpha_3_tip, beta_2_tip, beta_3_tip, U_tip, V_2_tip, C_2_tip, M_2_rel_tip, M_2_tip, M_3_rel_tip = aeroturbine.
                #incerased alpha_2 to 73 from 70
                if 40 < alpha_2 < 75  and U_hub < 335.28  and reaction_hub > 0 and M_2_rel_hub > M_2_rel_tip and 3150 < omega < 4500 and P_
                    # ANGLES CHECK
                    if 0 < alpha_2_tip < alpha_2 < alpha_2_hub and 0 < beta_2_tip < beta_2m < beta_2_hub and 0 < alpha_3_tip < k < alpha_3_

                        # OFF DESIGN CALCULATIONS
                        T_3_od,rho_3_od, P_3_od, alpha_3_od, beta_2_od, flow_coeff_2_od, incidence_2, v_2_od, C_w_3_od,C_a_3_od,U_mean_od,f
```

```
                    if flow_coeff_3_od > flow_coeff_2_od and C_a_3_od > 0 and 10 <= alpha_3_od <= 40 and P_02_rel_od > P_03_rel_od:

                        # VANE AREA AND HEIGHT
                        A_vane_mean = (A_1 + A_2)/2
                        h_vane_mean = (A_vane_mean * N/60)/U
                        r_tip_stator = r_meanline + (h_vane_mean/2)
                        r_hub_stator = r_meanline - (h_vane_mean/2)

                        # DESIGN POINT LOSSES
                        # CALCULATE LOSSES - Stator
                        K_p_stator, pitch_chord_ratio_stator, K_accel_stator, stagger_angle_stator, pitch_chord_ratio_stator, pitch_axi
                        K_s_stator = aerodynamic_losses.secondary_losses.calc_K_s(K_accel_stator, AR_vane, -10, alpha_2)
                        K_TET_stator, N_stator, c_true_stator, c_a_stator,throat_opening_stator = aerodynamic_losses.trailing_edge_loss
                        K_stator = K_p_stator + K_s_stator + K_TET_stator
                        pitch_stator = pitch_chord_ratio_stator*c_true_stator

                        # CALCULATE LOSSES - Rotor
                        K_p_rotor, pitch_chord_ratio_rotor, K_accel_rotor, stagger_angle_rotor, pitch_chord_ratio_rotor, pitch_axial_ch
                        K_s_rotor = aerodynamic_losses.secondary_losses.calc_K_s(K_accel_rotor, AR_rotor, beta_2m, beta_3m)
                        K_TET_rotor, N_rotor, c_true_rotor, c_a_rotor, throat_opening_rotor = aerodynamic_losses.trailing_edge_losses_r
                        K_rotor = K_p_rotor + K_s_rotor + K_TET_rotor
                        pitch_rotor = pitch_chord_ratio_rotor*c_true_rotor

                        # EFFICIENCY CALCULATIONS DESIGN POINT
                        eta_tt = aerodynamic_losses.efficiency_calculations(K_stator, K_rotor, M_2, M_3_rel, C_2, V_3)
                        delta_n, eta_final = aerodynamic_losses.efficiency_final(eta_tt, h, beta_3m, r_tip, r_meanline)

                        # OFF DESIGN LOSSES
                        eta_tt_od, delta_n_od, eta_final_od = aerodynamic_losses.losses_off_design(K_p_rotor, K_s_rotor, K_stator, K_1_

                        # EFFICIENCY CALCULATIONS OFF DESIGN
                        zxs = verify_zweifel_stator(r_hub_stator, h_vane_mean, N_stator, c_a_stator ,-10, alpha_2)
                        zxr = verify_zweifel_rotor(r_hub, h, N_rotor, c_a_rotor , beta_2m, beta_3m)

                        # WORK CHECKS
                        work_Part_A = c_p_gas * 1000 * (T_01-T_03)
                        work_check_cw = U * (C_w_2 + C_w_3)
                        work_check_vw = U * (V_w_2 + V_w_3)

                        if eta_final_od > 0 and zxs < 0.89 and zxr < 0.94: # This is to filter the abnormal behavior with the graph

                            # ================================================
                            # START OF LOOP 2
                            # THIS LOOP PROVIDES A NEW DESIGN POINT AND ALSO UPDATES THE VALUES FOR ROTOR INLET AND OUTLET RELATIVE FLOW
                            # Change the incidence angle
                            beta_2 = -1*incidence + beta_2m
                            beta_3 = beta_3m
                            incidence_off = incidence_2 - incidence

                            # OFF DESIGN LOSSES
                            eta_tt_od_new, delta_n_od_new, eta_final_od_new = aerodynamic_losses.losses_off_design(K_p_rotor, K_s_rotor

                            if eta_final_od_new > 0 and N_rotor < 50 and eta_final_od_new > 83:

                                eta_final = eta_final/100
                                eta_final_opt = 1/eta_final
                                eta_final_od = eta_final_od/100
                                eta_final_od_new = eta_final_od_new/100

                                delta_eta_optimize = eta_final - eta_final_od_new
                                data_scratch.append((zxs, zxr))
                                data_meanline.append((j, l ,flow_coefficient_2, flow_coefficient_3, omega, m,    # Non dimensional para
                                                      T_01, T_1, P_01, P_1, rho_1, T_02, T_2, P_02, P_02_rel, P_2, rho_2, T_03, T_3, P_03
                                                      M_1, C_1, C_a_1, C_w_1, -10, M_2, C_2, C_a_2,  C_w_2, alpha_2, M_2_rel, V_2, V_w_2,
                                # alpha_2_hub, alpha_3_hub, beta_2_hub, beta_3_hub, U_hub, V_2_hub, C_2_hub, M_2_rel_hub, M_2_hub, react
                                # alpha_2_tip, alpha_3_tip, beta_2_tip, beta_3_tip, U_tip, V_2_tip, C_2_tip, M_2_rel_tip, M_2_tip, M_3_
                                data_root_hub.append((reaction_hub, h, r_tip, U_tip, r_hub, r_tip_stator, r_hub_stator, h_vane_mean,
                                                      alpha_2_hub, alpha_3_hub, beta_2_hub, beta_3_hub, U_hub, V_2_hub, C_2_hub, M_2_rel_hu
                                                      alpha_2_tip, alpha_3_tip, beta_2_tip, beta_3_tip, U_tip, V_2_tip, C_2_tip, M_2_rel_ti

                                data_meanline_losses.append((c_true_stator, c_a_stator, stagger_angle_stator,K_p_stator, K_s_stator, K_

                                data_off_design.append((T_3_od, rho_3_od, P_3_od, alpha_3_od, beta_2_od, flow_coeff_2_od, incidence_2,

                                data_efficiency.append((eta_tt, eta_final, eta_final_od, eta_final_od_new, eta_final_opt, delta_eta_opt


    data_scratch = pd.DataFrame(data_scratch, columns=['zweifel_stator_NEW', 'zweifel_rotor_NEW'])
    data_meanline = pd.DataFrame(data_meanline, columns=['stage_loading', 'reaction_meanline', 'flow_coefficient_2', 'flow_coefficient_3',
    data_root_hub = pd.DataFrame(data_root_hub, columns=['reaction_hub', 'h', 'r_tip', 'U_tip','r_hub', 'r_tip_stator', 'r_hub_stator', 'h_
                                                         'alpha_2_hub', 'alpha_3_hub', 'beta_2_hub', 'beta_3_hub', 'U_hub', 'V_2_hub', 'C_2_
                                                         'alpha_2_tip', 'alpha_3_tip', 'beta_2_tip', 'beta_3_tip', 'U_tip', 'V_2_tip', 'C_2_
    data_meanline_losses = pd.DataFrame(data_meanline_losses, columns=['c_true_stator', 'c_a_stator','stagger_angle_stator','K_p_stator', '
    data_off_design = pd.DataFrame(data_off_design, columns=['T_3_od', 'rho_3_od', 'P_3_od', 'alpha_3_od', 'beta_2_od', 'flow_coeff_2_od',
    data_efficiency = pd.DataFrame(data_efficiency, columns=['eta_tt', 'eta_final', 'eta_final_od', 'eta_final_od_new', 'eta_final_opt','de

    return data_meanline, data_root_hub, data_meanline_losses, data_off_design, data_efficiency, data_scratch
```

run int main

In [ ]:
```
data_meanline, data_root_hub, data_meanline_losses, data_off_design, data_efficiency, data_scratch = int_main(range_mach_exit, range_stage_
```

Define Optimization Parameters

In [ ]:
```
data_efficiency['delta_eta_optimize_normalized'] = (data_efficiency['delta_eta_optimize'] - data_efficiency['delta_eta_optimize'].min()) /
data_meanline_losses['N_rotor_normalized'] = (data_meanline_losses['N_rotor'] - data_meanline_losses['N_rotor'].min()) / (data_meanline_loss
data_meanline['AN_squared_normalized'] = (data_meanline['AN_squared'] - data_meanline['AN_squared'].min()) / (data_meanline['AN_squared'].ma
data_efficiency['eta_opt'] = 1/data_efficiency['eta_final']
data_efficiency['eta_opt_normalized'] = (data_efficiency['eta_opt'] - data_efficiency['eta_opt'].min()) / (data_efficiency['eta_opt'].max()
```

In [ ]:
```
# OPTIMIZATION FUNCTION
data_efficiency['func_optimize'] = 0.4 * (data_efficiency['eta_opt_normalized']) + 0.3 * (data_meanline_losses['N_rotor_normalized']) + 0.2
```

Write results to file

In [ ]:
```
data_meanline.to_csv('_outputs/output_data_meanline.csv', index=False)
data_root_hub.to_csv('_outputs/output_data_root_hub.csv', index=False)
data_meanline_losses.to_csv('_outputs/output_data_meanline_losses.csv', index=False)
data_off_design.to_csv('_outputs/output_data_off_design.csv', index=False)
data_efficiency.to_csv('_outputs/output_data_efficiency.csv', index=False)
data_scratch.to_csv('_outputs/output_data_scratch.csv', index=False)
```